

AD-A059 603

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 17/7

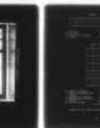
A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOT--ETC(U)

JUN 78 A L GONCALVES, J E CUBA BRAVO

UNCLASSIFIED

NL

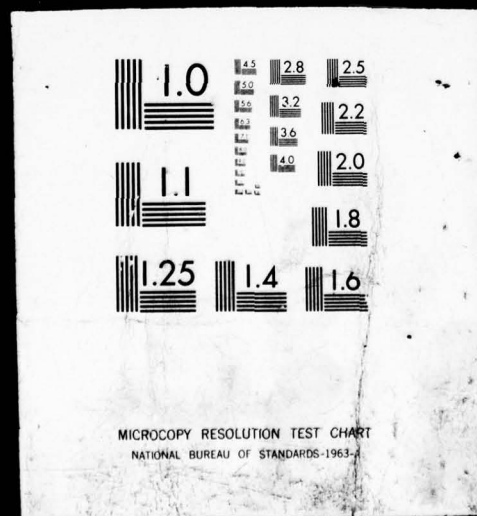
1 OF 6
AD
A059603



PRINTED

1 OF 6

AD
A 0 59603

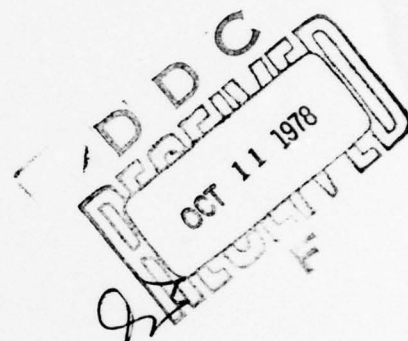


1
LEVEL

(2)

AD A059603

NAVAL POSTGRADUATE SCHOOL
Monterey, California



DDC FILE COPY

THESIS

(6) A MICROCOMPUTER BASED SHIPBOARD
SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM.

by

(10) Antonio Luiz Soares/Goncalves

and

Javier Enrique De la Cuba Bravo

(11) June 1978

(9) Master's thesis,

(12) 509p.

Thesis Advisor:

Stephen T. Holl

Approved for public release; distribution unlimited.

251 450

78 10 06 051

JOB

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Microcomputer Based Shipboard Surface-Sub- surface Contact Plotter System		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1978
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Antionio Luiz Soares Goncalves Javier Enrique De la Cuba Bravo		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1978
		13. NUMBER OF PAGES 508
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Microcomputer Plasma Display Microprocessor Shipboard Computer Maneuvering Board DRT Floating Point Hardware ASW		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) → Today, in most shipboard Combat Information Centers, true motion plots of own ship's motion and or surface/subsurface contacts are provided with the aid of electromechanical devices like the dead-reckoning tracer ("DRT") or the NC-2 plotter. These devices suffer a variety of drawbacks, such as inflexibility and the need to "track" the light-spots manually. This thesis has developed a flexible, labor-saving, true-motion plotter using contemporary microcomputer and plasma display technologies. An		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 68 IS OBSOLETE
S/N 0102-014-66011

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

CONT → automatic display of contact data, analogous to the plexiglass "surface status board" is also provided. Contact course, speed and CPA information are computed automatically and, unlike the DRT or the NC-2 plotter, both the geographic and tabular displays can be easily duplicated on the bridge and elsewhere.

ACCESSION for	
NTIS	Ville Section <input checked="" type="checkbox"/>
DDC	B.H. Section <input type="checkbox"/>
NAVATIONCO	<input type="checkbox"/>
J. & I. SECTION	
BY	
DISTRIBUTION/AVAILABILITY CODES	
01	02 03 04 SPECIAL
A	

DD Form 1473
1 Jan 73
S/N 0102-014-6601

UNCLASSIFIED

2 SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

Approved for public release; distribution unlimited.

A MICROCOMPUTER BASED SHIPBOARD
SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM

by

Antonio Luiz Soares Goncalves
Lieutenant Commander, Brazilian Navy
B.S., Brazilian Naval Academy, 1964

and

Javier Enrique De la Cuba Bravo
Lieutenant (JG), Peruvian Navy
B.S., Peruvian Naval Academy, 1974
M.S. Comp. Sys. Mgt., Naval Postgraduate School, 1977

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE, 1978

Authors:

Antonio Luiz Soares Goncalves
Javier Enrique De la Cuba Bravo

Approved by:

Stephen F. Hall Thesis Advisor
Mark Moraville Second Reader
Mr. H. H. H. Chairman, Department of Computer Science
A. Strady Dean of Information and Policy Sciences

ABSTRACT

Today, in most shipboard Combat Information Centers, true motion plots of own ship's motion and of surface/subsurface contacts are provided with the aid of electromechanical devices like the dead-reckoning tracer ("DRT") or the NC-2 plotter. These devices suffer a variety of drawbacks, such as inflexibility and the need to "track" the light-spots manually. This thesis has developed a flexible, labor-saving, true-motion plotter using contemporary microcomputer and plasma display technologies. An automatic display of contact data, analogous to the plexiglass "surface status board" is also provided. Contact course, speed and CPA information are computed automatically and, unlike the DRT or the NC-2 plotter, both the geographic and tabular displays can be easily duplicated on the bridge and elsewhere.

TABLE OF CONTENTS

LIST OF TABLES.....	9
LIST OF FIGURES.....	10
I. INTRODUCTION.....	11
II. INTRODUCTION TO THE PROBLEM.....	12
A. USUAL COMBAT INFORMATION CENTER OPERATIONS..	12
B. MANEUVERING BOARD PLOTTING SHEETS.....	12
C. DEAD-RECKONING EQUIPMENT.....	13
III. RECENT MICROPROCESSOR TECHNOLOGY ADVANCES.....	17
IV. CONCEPTUAL DESIGN.....	20
A. HARDWARE DESIGN CONSIDERATIONS.....	21
B. SOFTWARE DESIGN CONSIDERATIONS.....	22
1. Language Selection.....	22
2. Floating Point Arithmetic.....	22
3. Transcendental Functions.....	24
4. Design Philosophy.....	24
5. User Interface.....	27
6. Levels of Correctness.....	28
V. SYSTEM DESCRIPTION.....	31
A. HARDWARE DEPENDENCIES.....	31
B. SYSTEM CHARACTERISTICS.....	33
1. System Displays.....	35
a. Video Terminal Display.....	35
(1) Input Mode.....	40
(2) Display Mode.....	41
(3) Other Function-Defined Keys....	43

b. Plasma Display.....	43
C. DATA STRUCTURES.....	46
1. Data Structure for the System Parameters.....	48
2. Data Structures for Own Ship Parameters.....	50
3. Data Structures for the Contact Parameters.....	53
VI. CONCLUSION.....	58
A. RESULTS.....	58
B. FUTURE WORK.....	58
APPENDIX A ALGORITHM DESCRIPTION.....	60
A. THE CLOSEST POINT OF APPROACH (CPA).....	60
1. The Basic Relative Movement Problem.....	60
2. Other Uses of the Maneuvering Board.....	60
3. Maneuvering Board Problems Solutions Implemented.....	62
4. Problems that Normally Occur.....	62
5. The Least-squares Fit Approach.....	62
6. CPA Algorithm.....	65
B. GRAPHICS ON PLASMA DISPLAY.....	70
1. Physical Considerations.....	70
2. Plasma Display Unit Capabilities.....	70
3. Algorithm Design.....	71
a. Windowing.....	71
b. Procedure to Check if a Given Position Falls within the	

Limits of the Defined "Window".....	71
c. Normalization.....	73
d. Plasma Reorientation.....	74
C. TRANSCENDENTAL FUNCTIONS.....	75
1. Cosine and Sine Functions.....	76
2. Arc Tangent Function.....	77
D. POSITIONAL DATA CONVERSION.....	78
1. Convert LAT and LONG to X/Y Coordinates.....	79
2. Convert a Given Position in Terms of Bearing and Range from Own Ship to X/Y Coordinates.....	79
3. Convert X/Y Coordinates of a Given Position Into Latitude and Longitude....	80
APPENDIX B SOFTWARE CATEGORIZATION.....	81
A. MODULES DESCRIPTION.....	81
B. MODULES INTERACTION.....	81
APPENDIX C OPERATOR'S MANUAL.....	85
APPENDIX D FLOATING-POINT HARDWARE BOARD.....	134
A. GENERAL INFORMATION.....	134
B. DESCRIPTION OF THE MATH UNIT.....	134
C. PREPARATION FOR USE.....	136
1. Installation Considerations.....	136
2. I/O Base Address Switches.....	136
3. Programming Information.....	137
4. Math Unit Functions.....	139
5. Argument and Result Data Formats.....	139

6. Status and Flags.....	141
7. Examples of Floating-Point Number Representation.....	147
APPENDIX E PROGRAM LISTINGS.....	149
A. EXTERNAL DECLARATIONS.....	149
B. PROCEDURES BY MODULE.....	149
C. LISTINGS.....	156
LIST OF REFERENCES.....	505
INITIAL DISTRIBUTION LIST.....	507

LIST OF TABLES

I.	MATH UNIT FUNCTIONS AND EXECUTION TIMES.....	25
II.	FORMATS AND UNITS USED.....	36
III.	CONVERSION FACTORS USED.....	37
IV.	I/O ADDRESSING.....	138
V.	ARITHMETIC AND CONVERSION FORMATS.....	140
VI.	OPERATION ARGUMENT AND RESULT DATA FORMATS.....	142
VII.	FLAG BYTE FORMAT.....	143
VIII.	STATUS BYTE FORMAT.....	143
IX.	FLOATING-POINT NUMBERS.....	148

LIST OF FIGURES

1.	DRT BASIC COMPONENTS.....	14
2.	MK NC-2 MOD 2 PLOTTING SYSTEM.....	16
3.	EQUIPMENT CONFIGURATION.....	23
4.	VIDEO TERMINAL DISPLAY PICTURE ARRANGEMENT.....	38
5.	KEYBOARD LAYOUT.....	44
5a.	INPUT AND DISPLAY KEYS ARRANGEMENT.....	45
6.	PLASMA DISPLAY VIEW.....	47
7.	DATA STRUCTURE FOR THE SYSTEM PARAMETERS.....	49
8.	DATA STRUCTURES FOR OWN SHIP PARAMETERS.....	51
9.	DATA STRUCTURES FOR CONTACT PARAMETERS.....	54
10.	MANEUVERING BOARD SCHEMATIC.....	61
11.	USUAL PLOTTING AT THE MANEUVERING BOARD.....	63
12.	SPECIAL CASES.....	66
13.	WINDOWING SCHEMATIC.....	72
14.	EQUIPMENT VIEW.....	86

I. INTRODUCTION

A rapidly developing and expanding area of computer technology and application is that of microprocessors; paralleling the advances in microprocessor technology, the manufacturers of graphic displays are making steady progress in expanding their local capability, while at the same time reducing their cost.

Because a significant portion of the cost of building or maintaining a warship is the electronics in its sensor and weapons systems, and because a great amount of time and personnel power is employed in performing simple but important tasks, microcomputers offer the potential to: (1) reduce the cost of digital systems, (2) perform some complex functions at remote stations, relieving the congestion at larger central computing facilities, and (3) perform functions currently handled by watch personnel, thus reducing the manning requirements of watch sections. An example of this is the problem of manual tracking of radar contacts and the solution of Maneuvering Board problems.

It is the purpose of this thesis to demonstrate that an alternate approach to the solution of the problems mentioned above can be developed and implemented by using a microcomputer plasma graphics system, while at the same time maintaining a human engineered user interface.

II. INTRODUCTION TO THE PROBLEM

On board naval ships not equipped with Naval Tactical Data Systems (NTDS), operations performed by the Combat Information Center (CIC) during a normal peacetime watch include manual tracking of radar contacts and solution of maneuvering board problems.

A. USUAL COMBAT INFORMATION CENTER OPERATIONS

During normal peacetime steaming, the CIC watch team may consist of from two to ten or even more personnel, depending on the size of the ship as well as on the complexity of the equipment being used.

Among the problems that are normally solved by CIC personnel, special mention needs to be made of those of plotting contacts and the determination of parameters such as course, speed and closest point of approach (CPA) of those contacts. This is a tedious and error-prone task; it often requires most of the time and effort of the CIC team, and it is vitally important to the safety of the ship. This has lead to the installation of equipment to reduce the amount of workload in the CIC while at the same time improving the reliability of the constant information provided to the bridge; this equipment includes dead-reckoning devices and the NC2 plotter.

B. MANEUVERING BOARD PLOTTING SHEETS

The primary responsibility of CIC is to provide

information and recommendations on the tactical situation. Accordingly, CIC must supply information on all surface contacts within range. Contact course, speed and CPA information, is usually found using the "Maneuvering Board" plotting sheets.

The Maneuvering Board plotting sheet (H. O. 2665-10) has been prepared in order to facilitate the solution of a ship's relative movement problem.

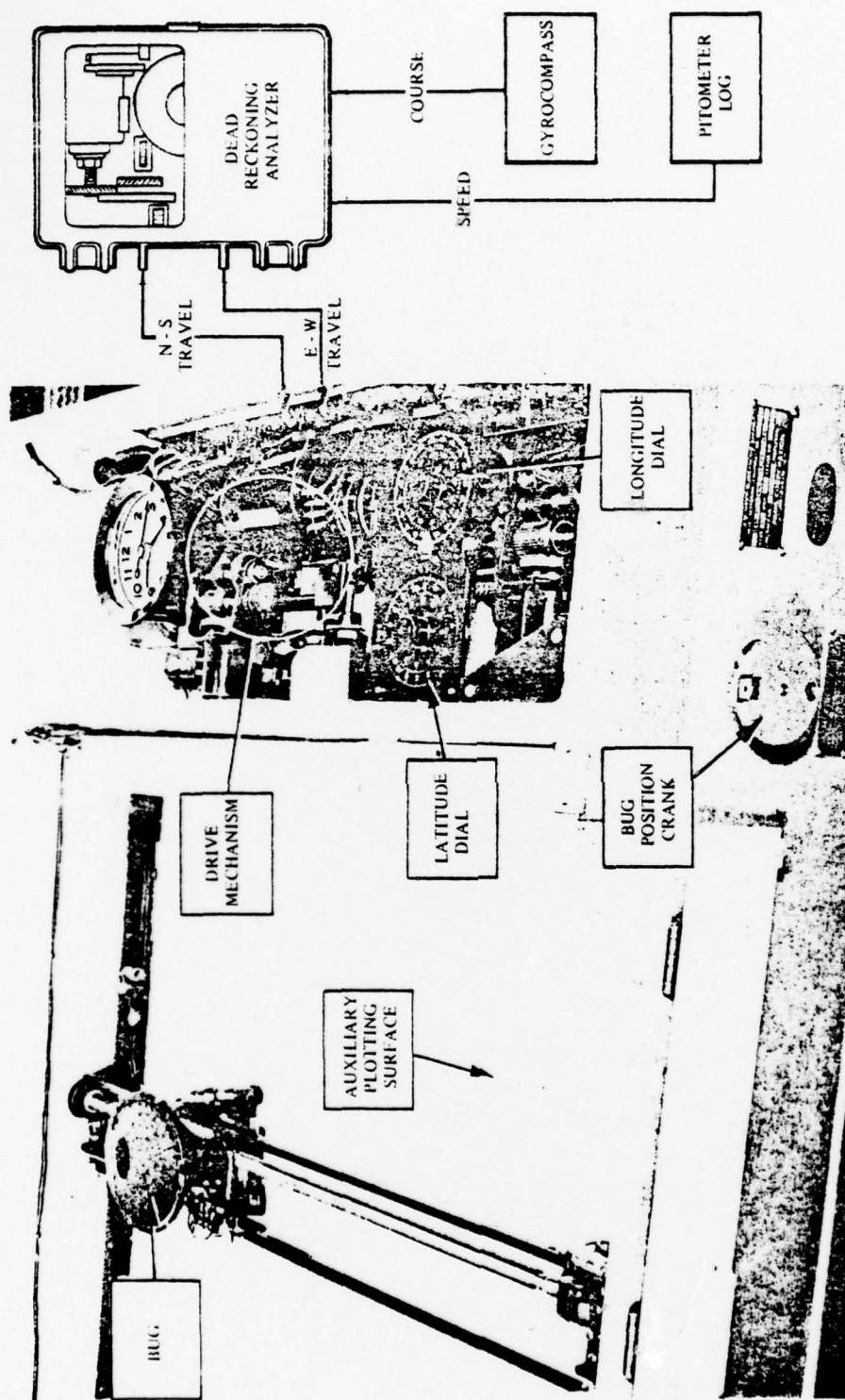
Although the use of the Maneuvering Board becomes after some practice, straightforward, it will normally require the complete attention of one person during CIC operations.

C. DEAD-RECKONING EQUIPMENT

Dead-reckoning equipment is an important device in CIC. It can maintain a continuous, up-to-the-minute, geographic plot of own ship's motion.

The dead-reckoning system consists of the following basic components: (1) dead-reckoning analyzer (DRA), (2) dead-reckoning indicator (DRI), and (3) dead-reckoning trace (DRT), which are shown in Figure No. 1. Course and speed inputs from the own ship are fed into the DRA from the gyrocompass and pitometer log, and then to the DRT, where they cause a movable source of light to trace the ship's track continuously.

The chief value of the DRT is its use in analyzing ship movements and in planning and carrying out maneuvers. As a geographic plotting device, the DRT displays true courses and allows direct computation of true contact speeds.



B (DRA)

A (DRT)

FIGURE NO. 1 DRT BASIC COMPONENTS

Marking positions of the bug indicates true positions of own ship; connecting these plotted positions yields the ship's track. Plotting ranges and bearings of contacts, using own ship's positions as references, establishes their true positions. An experienced DRT operator can maintain simultaneous plots of as many as half a dozen contacts, meanwhile supplying essential data (as required) on contacts that are being plotted.

A problem that is present, especially in Anti Submarine Warfare (ASW), is that of plotting fast moving ships, and tight-turning submarines, which normally results in confused and inaccurate DRT plots. In order to help solve these problems, as well as to reduce inconvenient delays in plotting contacts because of information relayed through phone talkers, the USN Mk NC2 plotting system (Canadian DRT) was developed. The NC2 plotter consists of three major units: (1) the plotting table, (2) the dead-reckoning indicator, and (3) data converter.

The main difference between the NC2 plotting system and the DRT is that the NC2 plotting system is capable of receiving contact bearing and range information directly from four sources (usually 3 radar repeaters and the sonar). This information is then translated, and presented as colored points of light in the plotting table. A functional diagram of the NC2 plotting system is presented in Figure No. 2.

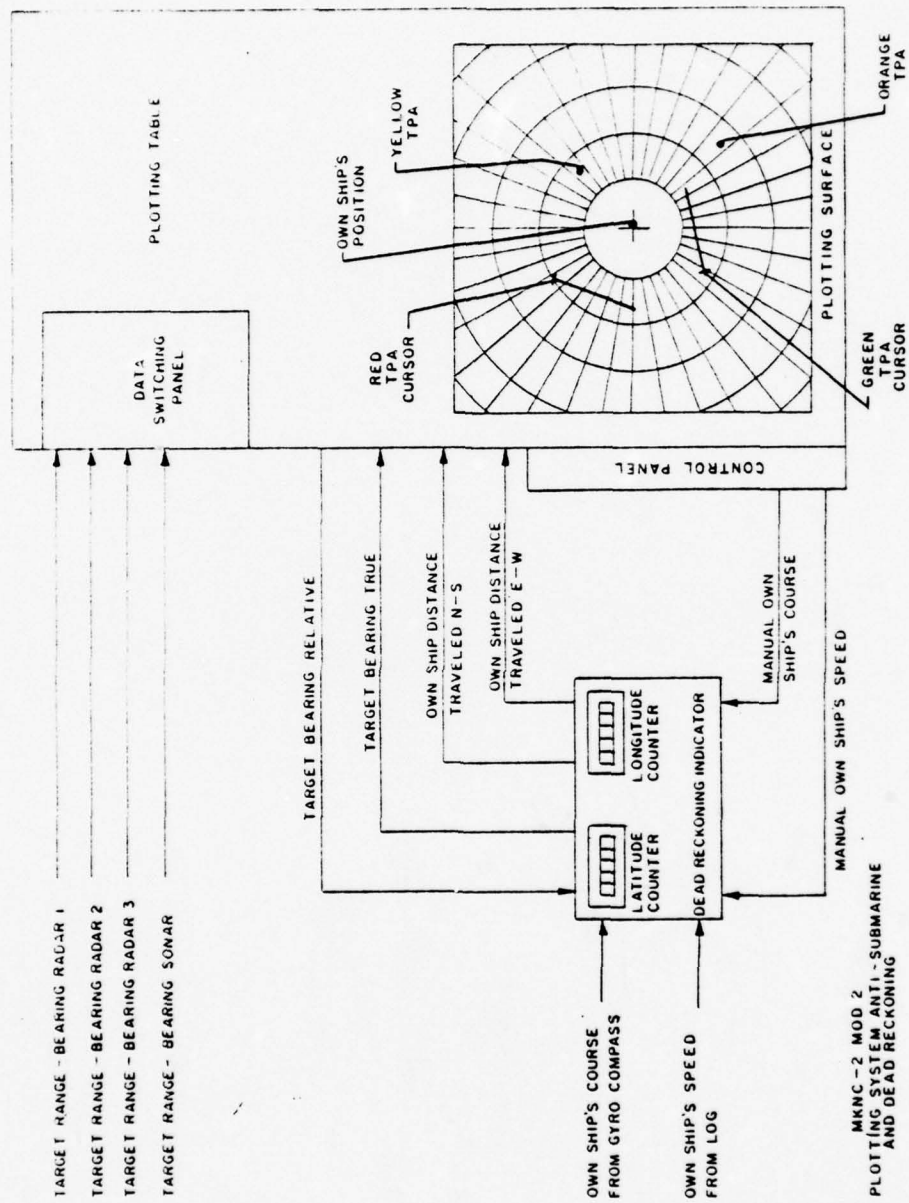


FIGURE NO. 2 MK NC-2 MOD 2 PLOTTING SYSTEM

III. RECENT MICROPROCESSOR TECHNOLOGY ADVANCES

Recent advances in the field of large-scale integration (LSI) semiconductor process technology have made possible substantial reductions in the cost and size of digital logic circuits. In the last decade, computer system building blocks have progressed from discrete components to complex integrated circuits. Microprocessors represent a very remarkable achievement of engineering ingenuity and industrial know-how at their best.

Composed of miniaturized software-encoded chips and minimal hardware circuitry, the microcomputer, a digital computer using microprocessor logic, is ideally adapted to specialized tasks and applications.

Since 1973, the year in which INTEL Corporation shipped the first 8080 8-bit N-channel microprocessor, a number of manufacturers have developed similar products and because of this competition prices have been drastically lowered. With hardware and software system development costs as they are, much micro work is done on a custom basis. The greatest advantage of microcomputer based systems is the ability to have specialized and dedicated equipment solving specific problems.

At the same time peripheral equipment has also experienced major changes, perhaps most dramatically in graphic displays. The cathode ray tube (CRT) is the most common device because of its high performance, low cost, and

quality.

A relatively new graphic device is the Plasma Display. A plasma panel is two etched glass plates separated by a neon-based gas which glows when excited by an electric pulse. The display consists of a series of bright dots that can be formatted into alphanumeric and graphic symbols. Plasma panels do not require refresh procedures and once a particular point on the display is "turned on", it continues to glow until "turned off".

The simplicity of construction of the plasma panel suggests that it can potentially replace the CRT for many computer graphic applications. In its current state of development, it presents the following advantages: (1) it presents a sharper image that does not deteriorate with time (does not need to be refreshed), (2) it has a reliable selective erasure mechanism, (3) its power requirements are comparatively low, (4) it has a longer expected life time, and (5) it occupies less space. Some disadvantages are: (1) lower resolution, (2) relatively slow write and erase rates, and (3) no "real" gray scale [Ref 11].

As O. Babin and R. Seaman have shown [Ref. 1], it is possible to combine the advantages and convenience of low-cost microcomputer systems, with the powerful technology of plasma displays, thus having a complete microcomputer development system with graphics capability.

The purpose of this thesis was to demonstrate the applicability of a simple and inexpensive microcomputer

plasma graphics system to Navy functions and problems, specifically as an alternate approach to the solution of CIC's most common problems, and to show that it is possible to design and develop a human engineered user interface.

IV. CONCEPTUAL DESIGN

Microprocessor technology can be used to solve Maneuvering Board problems and to present a geographical plot of own ship and contact positions.

In order for such a system to be competitive with existing equipment, the following requirements should be fulfilled:

1. Relatively inexpensive acquisition and maintenance costs.
2. Reliable and widely available components.
3. Speed and accuracy in performing necessary calculations.
4. Human engineered user interface.
5. Flexibility in its use.
6. Capacity of displaying a geographic plot of the own ship and contacts, while maintaining at the same time, a constant update of the own ship's position.
7. Capacity to solve Maneuvering Board problems such as the determination of CPA information, and calculation of course and speed values for the contacts.
8. Operation of the proposed system should not require more people than the current methods.

It was with these objectives in mind that the microcomputer system described below was designed.

A. HARDWARE DESIGN CONSIDERATIONS

The equipment selected to implement the system described in this thesis was chosen because it was available at the Naval Postgraduate School, and was representative of commercially available microcomputers and plasma display technologies. The selection of the INTEL Microcomputer Development System (MDS) computer as the prototype for this project was made because of its immediate availability at the Naval Postgraduate School and because O. Babin and R. Seaman [Ref 1] had already interfaced the MDS with a plasma display (the AN/UYO-10 Plasma Display Set, manufactured by SCIENCE APPLICATIONS INC (SAI)).

Two features of the MDS system that were especially useful were: (1) the 8-level, nested interrupt priority resolution network, and (2) the real-time clock logic, used to maintain a real time clock value by means of generating an interrupt at 0.77 millisecond intervals.

A DATAMEDIA Elite 2500 Video Terminal was chosen to present alphanumeric information, because its features included: (1) editing and roll operation modes, (2) 50 to 9600 baud programmable speed transmission, (3) protected fields, (4) computer derived or high light field (blink), (5) addressable cursor, and (6) provision to drive up to 16 external monitors.

Because of the volume of floating point calculations required, an SBC 310 High Speed Mathematics Unit, developed by Intel Corporation, was incorporated. In performing high-

speed mathematical functions, the Math Unit acts as an intelligent processor, performing a repertoire of up to 14 arithmetic functions at least an order of magnitude faster than comparable software routines.

Figure No. 3 shows the final configuration of the equipment used in developing this thesis.

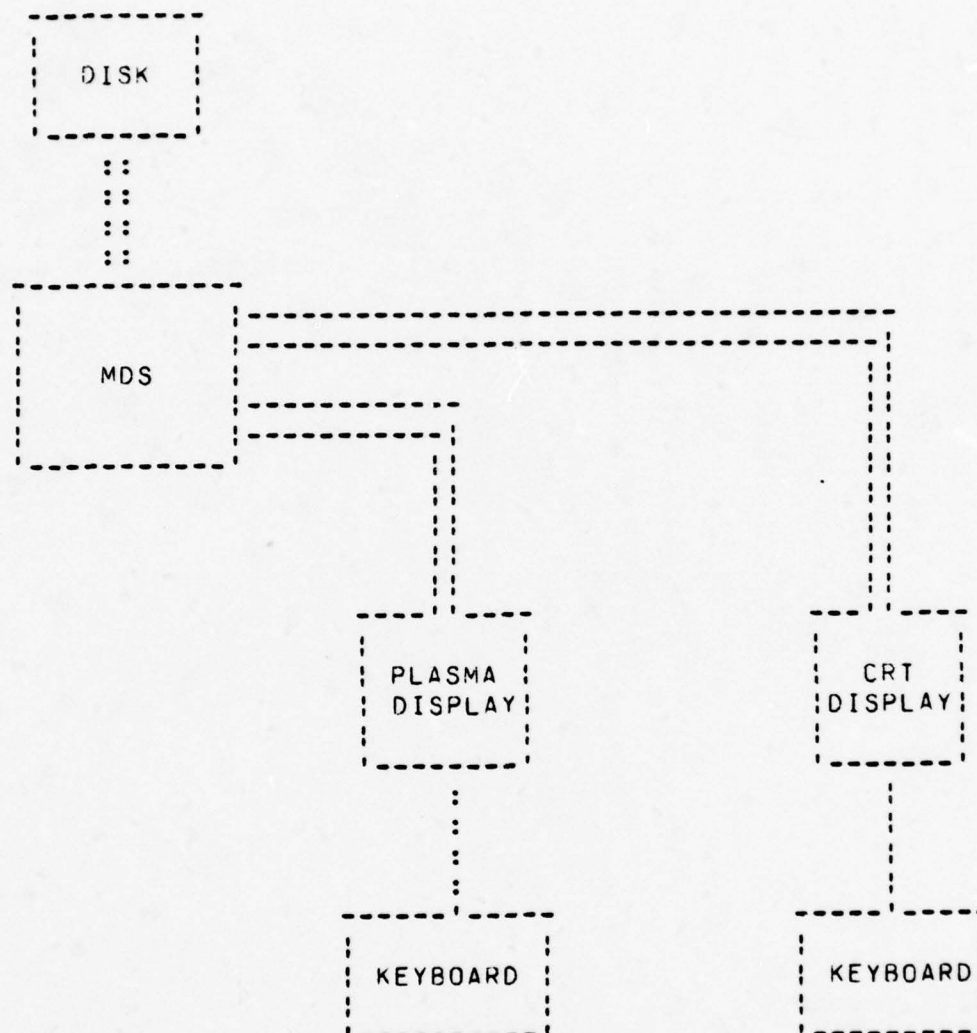
B. SOFTWARE DESIGN CONSIDERATIONS

1. Language Selection

Because of the hardware configuration selected and its immediate availability, PL/M 80 was chosen as the programming language to be used. PL/M 80 is a language developed by Intel Corporation and designed especially for system and applications programming for the Intel 8080 microprocessor. The ISIS-II disk operating system, also developed by Intel Corporation for the Intellec MDS system, has a resident PL/M 80 compiler, which was very useful during the implementation, debugging and testing phases.

2. Floating Point Arithmetic

Floating point arithmetic was required because the range of numbers to be represented was large and sometimes unpredictable. The format used to represent floating point numbers was as required by the SBC 310 High Speed Mathematics Unit. Appendix D shows how this unit was actually implemented and the required software procedures that are needed to use it. The functions that were



NOTE.- denotes connection used only for implementation purposes.

FIGURE NO. 3 EQUIPMENT CONFIGURATION

implemented and the execution times are listed in Table I.

3. Transcendental Functions

Three transcendental functions were also implemented, namely: (1) cosine of a given angle in radians, (2) sine of a given angle in radians, and (3) arc tangent of the ratio of two input parameters. These functions were implemented based on the procedures in Reference 4, using floating point procedures. Appendix A describes the algorithms used.

4. Design Philosophy

The system was designed with the following three specific objectives in mind:

- a. Human engineered user interface.
- b. Capability to display a geographic plot of the own ship and contact positions.
- c. Capability of solving Maneuvering Board problems, specifically the determination of CPA information, course and speed values of all contacts.

In order to achieve these objectives, a bottom-up implementation philosophy was chosen. Because of the modular design encouraged by PL/M 80 and ISIS-II, it was also possible to map the different levels of design into corresponding modules of software. The basic idea was to encompass all functions corresponding to a level of design into one software module capable of performing all the necessary functions.

In doing so, the following modules were developed:-

OPERATION NAME	OPERATION CODE	TYPICAL EXECUTION TIME	MAXIMUM EXECUTION TIME
Fixed Point Multiply (MUL)	0	15	20
Fixed Point Divide (DIV)	1	26	30
Extended Fixed Point Div(EDIV)	E	84	100
Float. Point Multiply (FMUL)	2	84	100
Float. Point Divide (FDIV)	3	92	110
Float. Point Add (FADD)	4	33	75
Float. Point Subtract (FSUB)	5	33	75
Float. Point Square (FSQR)	6	84	100
Float. Point Sqr Root(FSQRT)	7	178	205
Fixed-to-Float Cnvrson(FLTDS)	8	72	100
Float-to-Fixed Cnvrson(FIXSD)	9	42	85
Float. Point Compare (FCMPR)	A	7	7
Float. Point Test (FZIST)	B	7	7
Exchange (EXCH)	F	4	4

Note: all time values are specified in microseconds.
 Listed times do not include time to pass arguments to
 the Math Unit and to read results upon completion: this
 is typically 90 microseconds.

TABLE I. MATH UNIT FUNCTIONS AND EXECUTION TIMES

a. BASICS: a module used to interface the system with the CRT/keyboard.

b. PLASMA\$PRIMITIVES: a module used to interface the system with the Plasma Display unit.

c. TIME: a module used to perform all functions dealing with time, and also to keep a real time clock for the system.

d. FLOATING\$POINT: a module used to perform all the necessary floating point operations, and to calculate the transcendental functions already described.

e. FLTASCII: a module used to make the necessary conversions from a string of ASCII characters into floating point format, and vice versa.

f. CRT: a module used to display all the necessary and requested values in the CRT.

g. PLASMA\$MODULE: a module used to display all the necessary graphic information in the Plasma Display unit.

h. COMMANDS: a module used to interface with the user to get requested input values.

i. DISPLAY\$CMDS: a module used to display information requested by the user, in the CRT.

j. CPA\$MODULE: a module used to calculate and solve all CPA values and problems.

k. EXECUTIVE\$CMDS: a module containing all the necessary procedures to link all the other modules' actions.

l. MAIN: a module used to serve as executive module of the system by combining all the different functions of

the already described modules.

Appendix B describes all these modules in more detail, while Appendix E lists all the programs that compose the system; Appendix A describes all the main algorithms used in the system, and Appendix C contains an Operator's Manual giving instructions on how to use the system.

5. User Interface

This is the aspect in which most systems fail, because of the complexity of the problem of trying to define what an "average user" is in any system, and in trying to encompass all the possible ways in which a user could react. It is therefore necessary to define boundaries in how the system is expected to interact with the user, while at the same time providing an acceptable range of variations.

One of the features that becomes indispensable when interacting with a human operator, is error detection and notification as soon as possible, and the capability of allowing the user to correct his own mistakes either when so told by the system, or when he discovers the mistake by himself. Special care was taken in this aspect, as is explained below.

One decision that was taken during system design was to reduce to a minimum the number of keys that the operator would have to press when entering data. This implied that no special character, such as carriage return, would be necessary to mark the end of an input, but this also meant at the same time that the input would have to be done in a

pre-formatted manner. The design choice of doing all inputs in a pre-formatted way, was reinforced by the fact that personnel in a CIC team are used to communicate with each other using a pre-defined terminology. That is, if the value of the course of a contact is 030 degrees, then when communicating this value, the word "thirty" is not used but instead the message "zero", "three", "zero" is given.

6. Levels of Correctness

As was established before, it is very important for a system that depends almost completely on the correctness of the input values that it receives, to ensure that all the input values received fall into an acceptable range of variation, and within logical and plausible limits; furthermore, the capability of error correction should also be embedded in the system.

To achieve all of this, the system was designed to have up to six levels of error detection and correction:

a. The first level is established by checking for invalid requests from the user, such as asking for information about a contact when no contacts are in the system; if this error occurs, a warning message is issued.

b. The second level of error detection is done by checking each character received from the keyboard, to determine if it has some meaning to the system. An example of this would be when trying to input an alphabetic character in a numeric defined field; if an error of this type is detected, then the CRT's alarm will sound and the

character will not be echoed at the display.

c. The third level is established by checking a syntactically correct input against established boundaries for the type of input being expected. An example of this would be when a value of 79 is received when requesting an HOURS value; since 79 lies outside the boundaries established for HOURS ($0 \leq \text{HOURS} < 24$) then an error has occurred; thus, a warning message will be issued, the CRT's alarm will sound, the value will not be accepted nor processed by the system, and the cursor will be placed at the beginning of the incorrect value; the warning message will remain on the screen until the mistake is corrected.

d. The fourth level occurs after all input has been received from the keyboard, by giving the user the chance to correct any value just input.

e. The fifth level is obtained by making the system do all the necessary prompting in a pre-determined format. This eliminated the problems that occur when more or less data than necessary is provided to the system.

f. The sixth level is carried out by allowing the user to change any current own ship or contact value at any time.

All these levels of correctness are ensured any time an input operation takes place. It is important to note that no provision for values that are syntactically correct and within established boundaries, but incompatible with previous data and logically incorrect with the present

situation, could be provided because of the many parameters, variables and special situations that could be involved at any given time.

Obviously, the use of so many checking procedures added a considerable amount of overhead to the system, as far as amount of code is concerned; however, sufficient core was available, and execution speed was not significantly degraded.

V. SYSTEM DESCRIPTION

The description of the system is divided into three major areas: hardware dependencies, system characteristics, and data structures.

A. HARDWARE DEPENDENCIES

Because of the hardware equipment that was selected the following hardware dependencies exist in the current implementation of the system:

1. The system utilizes the real time clock logic as provided by the MDS system; this clock is capable of generating an interrupt of level 1, when enabled, at fixed intervals of 0.77 milliseconds. In order to be able to use this feature, a procedure named CLOCK was implemented and defined to be of type INTERRUPT 7; this allowed the PL/M 80 compiler to create the necessary code for the interrupt vector and for the routine CLOCK. A software problem needs to be explained at this point; since the development of this system was done under ISIS-II, the CLOCK procedure had to be defined as if occurring at a level other than 0, 1 or 2 (in this case 7), because ISIS-II would not allow any user generated code to be located below memory location 3000H, except for code to be used in interrupts 3 through 7, or locations 24 to 63; since the clock interrupts are of level 1 (locations 8 through 15), then in order to override this ISIS-II inconvenience, the interrupt vector generated for

CLOCK, as starting in location 56 (level 7), had to be moved to location 8 (level 1), after locating the system code in memory, and before attempting to use the real time clock; References 6, 7 and 9, describe this problem in more detail.

2. The system occupies approximately 40K bytes of physical memory for code, and approximately 12K bytes to be used for variable data; therefore a configuration of at least 52K bytes of RAM is necessary to execute the system.

3. The system utilizes an SBC 310 High Speed Mathematics Unit to perform floating point arithmetic. Although the presence of this Math Unit could be avoided by replacing its functions with appropriate software routines performing the same operations using the same formats, this is not recommended because of the excessive overhead that would result, especially with regard to execution time. Appendix D explains how the Math Unit was actually implemented.

4. The system depends in three ways on the type of terminal used: the handshaking procedure necessary to communicate between the CPU and the terminal, the code needed to control the CRT's functions, and the general features of the DATAMEDIA Elite 2500 Video Terminal - notably the programmable roll mode, the setting of privileged fields, the capability of having an addressable cursor, and the possibility of making displayed messages blink.

5. Finally, the system has a hardware dependency with

respect to the Plasma Display Unit selected, in three respects: the handshaking procedure necessary to send characters to the Plasma Display Unit; the code used to control the Plasma Display Unit functions; and the capability of the Plasma Display Unit of working either in alphanumeric or in vector mode. The SAI Plasma Display Unit has a built-in capacity to draw solid or dashed vectors by specifying the two end points defining the vector.

B. SYSTEM CHARACTERISTICS

As was established before, the system was designed to perform basically the same functions as Dead-reckoning equipment, while adding the capability of simultaneously solving maneuvering board problems. The system, as implemented in this thesis, is capable of performing the following tasks:

1. Maintain as many as 30 positions of the own ship; maintain as many as 15 positions for each contact, for as many as 15 different contacts.
2. Maintain and present a geographic plot of own ship and contacts.
3. Maintain and display a Surface Status Board.
4. Solve the following Maneuvering Board problems: (a) CPA information, and (b) course and speed of contacts.
5. At user's request, prompt for necessary inputs and update the displays, if applicable.
6. At user's request, display all the information that

exists in the system, in a pre-established format.

7. Update automatically the position of the own ship, by using its course and speed values with a frequency specified by the user.
8. Maintain a Real Time Clock.

All the above tasks are performed by the system, either automatically or at user's request. Some parameters need to be defined during system initialization; these parameters are:

1. Time zone number.
2. Local time at which the system is started.
3. Latitude and longitude values defining a selected geographical point to be used as center of a Coordinated Grid system used mainly for plotting purposes and as a reference to determine positions of the own ship and contacts.
4. Latitude and longitude values defining the starting position of the own ship.
5. Initial course value of the own ship.
6. Initial speed value of the own ship.
7. Initial scale value for the Plasma Display Unit.

The system also has two parameters with initial default values: (1) the Safe CPA Range set at 50 yards, and (2) the interval of time between updates of the own ship's positions is set at 180 seconds. Any parameter in the system can be changed at any time, with the exception of those parameters that define the boundaries of work of the system; these

fixed values are:

1. Maximum range: 100.0 miles.
2. Maximum speed: 99.9 knots.
3. Number of letters used to designate a contact: 2.
4. Minimum scale value: 00.25 miles/inch.
5. Maximum scale value: 25.00 miles/inch.
6. Zero defined for the system:
- 0.0000009 <= "zero" <= +0.0000009 (For floating point numbers only.)
7. Minimum Safe CPA Range value: 50 yards.
8. Maximum Safe CPA Range value: 1000 yards.

Table II describes the formats used for input, internal, and output representation of the values with which the system operates, as well as the units used; Table III gives the conversion factors used in the system.

1. System Displays

As was stated previously, the system maintains two different displays: a Video Terminal presents a Surface Status Board and interactions with the user, and a Plasma Display presents a geographic plot of the positions of own ship and contacts.

a. Video Terminal Display

Figure No. 4 presents a picture of how the Video Terminal Display is arranged. The screen is divided in two areas; the upper portion of the screen consists of a fixed format presentation of information about the own ship and some contacts. From this representation, the following

VALUE	FORMAT		UNITS	
	I/O	INTERNAL	I/O	INTERNAL
TIME	xx:xx:xx	3 bytes	hrs : min : sec	same
COURSE	xxx.x	F.P.	degrees	same
SPEED	xx.x	F.P.	knots	same
BEARING	xxx.x	F.P.	degrees	same
RANGE	xxx.x	F.P.	miles	miles
	xxxxxx		yards	
SCALE	xx.xx	F.P.	miles/inch	same
LAT.	xx:xx.x	F.P.	degrees:minutes	minutes
LONG.	xxx:xx.x	F.P.	degrees:minutes	minutes
DESIG	Aa	Address	ASCII characters	decimal value

x Numeric character.
 F.P. Floating point representation.
 A Alphabetic character including space.
 a Alphabetic character excluding space.

TABLE II. FORMATS AND UNITS USED

1 nautical mile 2025.3716 yards.
 1 degree 0.0174532925 radians.
 1 minute: 0.00029089 radians.
 1 minute of long. ... 1 nautical mile.
 1 minute of lat. 1 nautical mile x Cos(Latitude).
 PI 3.141593
 1 knot 1 nautical mile / hour.

TABLE III. CONVERSION FACTORS USED

information can be obtained at any time:

(1) Time. The time zone number and the local time maintained and automatically updated by the system every second.

(2) Own Ship. Latitude and longitude values indicating its last geographical position as determined automatically by the system at least every T time, where T is a period of time as selected by the user. Information about the course and speed is also available.

(3) System. Information about the total number of contacts in the system, classified by their corresponding class: Friendly (F), Hostile (H), and Unknown (U). Information about the mode in which the system is operating is also displayed. This will be explained in the following paragraphs.

(4) Contacts. Complete information about six contacts is displayed. The following items are provided for each contact: (a) designation, (b) type: Surface (S) or Sub-surface (SS), (c) class: Friendly (FRI), Hostile (HOS), or Unknown (UNK), (d) last mark: time, bearing and range, (e) course, if known, (f) speed, if known, (g) CPA information, if known: time, bearing and range, or one of the following three possible messages: "timeCOLLISION", "SAME CRS & SPD", and "MOVING AWAY".

It should be noted that, although the system is capable of maintaining up to 15 contacts, information about only six of them will be constantly present at the display;

the user has the capability of selecting which contacts he desires to be displayed in this way, or as will be explained later, he can also obtain information about any contact (temporarily) in the lower portion of the screen.

The lower portion of the screen consists of the last eight rows and is used during Input and Display operations; it has no fixed format and if the system is not executing an Input or Display operation, it contains only the prompt ("%") symbol displayed in its upper left corner.

As mentioned previously, the system operates in three different modes; these modes are: (1) Initialization, (2) Input, and (3) Display. The Initialization mode is required only once at the beginning of the execution of the system, and it is indispensable for the operation of the system, as was explained previously; it can not be requested by the user once the system is operating. The Input and Display modes are determined by the way the system is performing at any given time. The system operates in the Input mode by default. Both modes require interaction with the user.

(1) Input Mode. The system is operating in Input mode any time an Input operation is being performed. The following Input operations can be requested by the user:

(a) Modify Coordinate Grid Origin parameters; optional: latitude and longitude.

(b) Modify Own Ship parameters; optional: latitude, longitude, course and speed.

(c) Create a Contact; required: designation, type, class, bearing, and range; optional: course and speed.

(d) Remove a contact; required: designation.

(e) Redesignate a Contact; required: old and new designations.

(f) Update a Contact; required: designation; optional: type, class, bearing, range, course and speed.

(g) Contacts to Display: select which contacts are desired to be displayed permanently in the Surface Status Board.

(h) Time; optional: time zone value, system clock value, and time between updates.

(i) Safe CPA Range: modify the Safe CPA Range parameter.

(j) Wind: enter/modify wind parameters; required: direction and speed.

(k) Scale: modify the graphics scale value.

(l) Plasma Reorientation: reorient the picture displayed at the Plasma Display Unit.

All these operations are performed using the lower portion of the screen and are divided into various phases ("pages") in order to allow a better utilization of the screen; any Input operation can be requested by pressing the appropriate key at the keyboard.

(2) Display Mode. The system is operating in Display mode any time a Display operation is being performed. The following Display operations can be requested by the user:

(a) Origin: display information about the Coordinate Grid Origin parameters (latitude and longitude).

(b) Scale: displays information about the Graphics Scale currently used by the system.

(c) Own Ship: displays information about the following own ship parameters: latitude, longitude, X and Y values in the user defined Coordinate Grid System being used, course and speed.

(d) Contact Information: displays information about a specified contact's parameters; requires: designation; provides: type, class, number of positions maintained by the system, latitude, longitude, X and Y values in the user defined Coordinate Grid System being used, last mark's time, bearing and range, and if possible, gives information about course, speed, CPA parameters, and estimated actual position.

(e) Contacts in System: displays information about the designations of all the contacts in the system, if any.

(f) Safe CPA Range: displays the value of the Safe CPA Range parameter.

(g) Wind: displays information about the wind.

(h) Display Update Time: displays the value of the current Time Between Updates being used.

All these operations are performed using the lower portion of the screen, and are divided into various phases ("pages") in order to allow a better utilization of the screen. Any Display operation can be requested by pressing the appropriate key at the keyboard.

Figure No. 4 presents a view of how the Video Terminal display looks during system operation; Figure No. 5 presents a layout of the keyboard used and Figure No. 5a the arrangement of the various Input and Display keys.

(3) Other Function-Defined Keys. The keyboard has also two other special function keys:

(a) Rubout Key. Used to backspace the cursor when inputting information into the system.

(b) "GO" key. Used to advance the various "pages" in which the Display operations are divided.

b. Plasma Display

The Plasma Display is used to present a geographical plot of the own ship and contact positions. It displays the geographical picture that is defined through a system-defined, and user-controllable "window"; this window is used to focus on the geographical area that is of interest to the user. Two mechanisms control the "window":

(1) Scale. This defines the scale at which the plotting is desired to be presented, determining the size of the window.

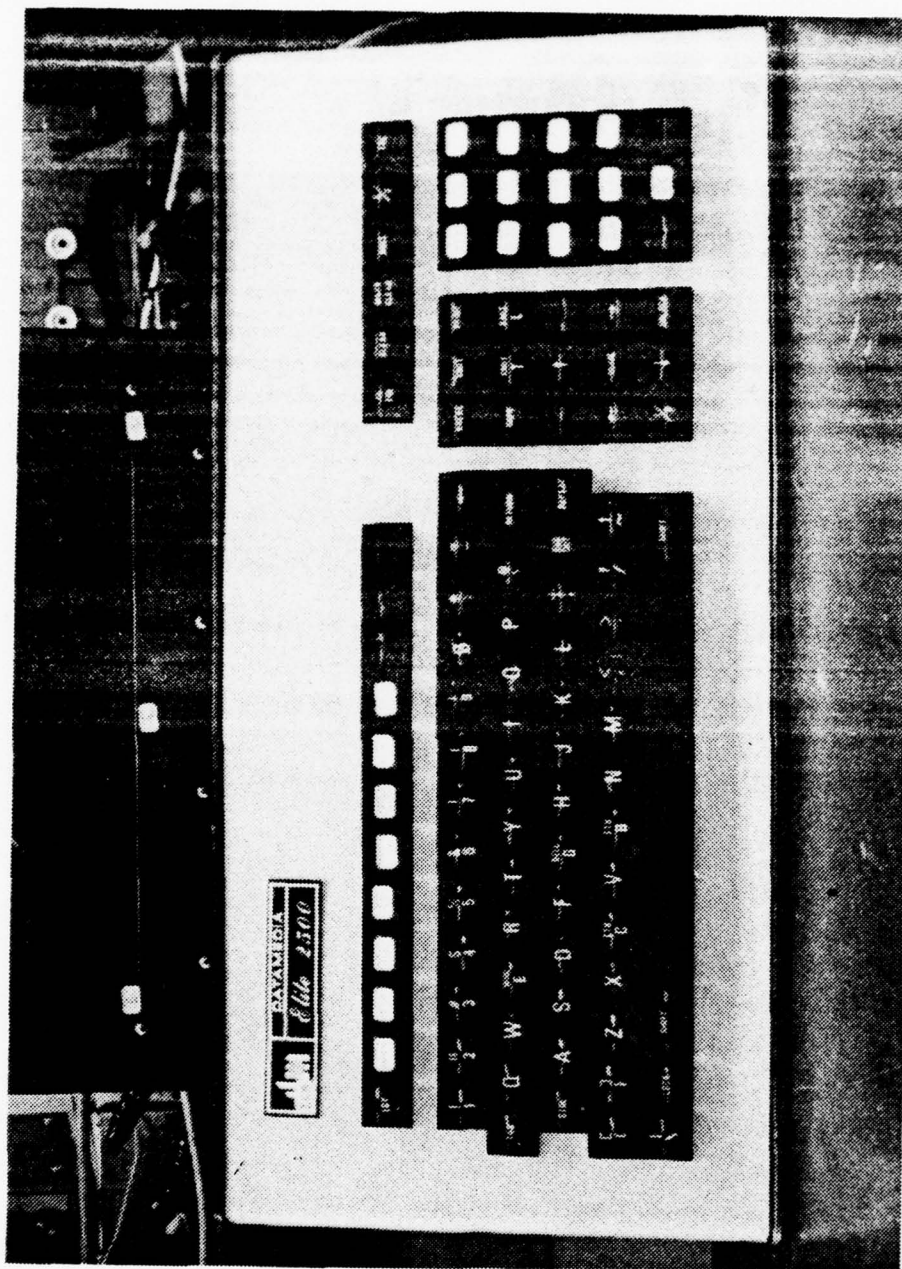


FIGURE NO. 5 KEYBOARD LAYOUT

DISPLAY KEYS

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

- | | |
|------------------------|-------------------------|
| 1. Origin. | 2. Scale. |
| 3. Own Ship. | 4. Contact Information. |
| 5. Contacts in System. | 6. Safe CPA Range. |
| 7. Wind. | 8. Display Update Time. |

INPUT KEYS

1	2	3
4	5	6
7	8	9
10	11	12
	13	

- | | |
|-----------------------------------|---------------------------|
| 1. Create a Contact. | 2. Modify Own Ship. |
| 3. Modify Coordinate Grid Origin. | 4. Remove a Contact. |
| 5. Contacts to Display. | 6. Wind. |
| 7. Redesignate a Contact. | 8. Time. |
| 9. Scale. | 10. Update a Contact. |
| 11. Safe CPA Range. | 12. Plasma Reorientation. |
| | 13. GO key. |

FIGURE NO. 5a INPUT AND DISPLAY KEYS ARRANGEMENT

(2) Picture Reorientation. This specifies the window's center. Three different methods were used: (a) fixed reorientation, by making as new center any one of eight (8) pre-defined points in the picture, (b) by making the last position of the own ship to be the new center, and (c) by making the last position of any contact to be the new center of the picture.

The Plasma Display also presents the current value of the scale being used in its upper left corner. The positions of the own ship are marked with bright circles connected by solid vectors, while the positions of the contacts are connected by dashed vectors and marked with two different symbols: a cross if the contact is Hostile or Unknown, and a circle if the contact is Friendly; the designation of any contact plotted at the Plasma Display is presented close to its first plotted position.

Figure No. 6 presents a view of how the Plasma Display appears during system operation; Appendix A gives the algorithms used for establishing the window and for forming the picture to be presented.

C. DATA STRUCTURES

The main data structures used in the system can be classified in three categories: (1) data structure used to represent system parameters, (2) data structures used to represent the own ship information, and (3) data structures used to represent contacts information. The main type of data structure used was the STRUCTURE, as provided

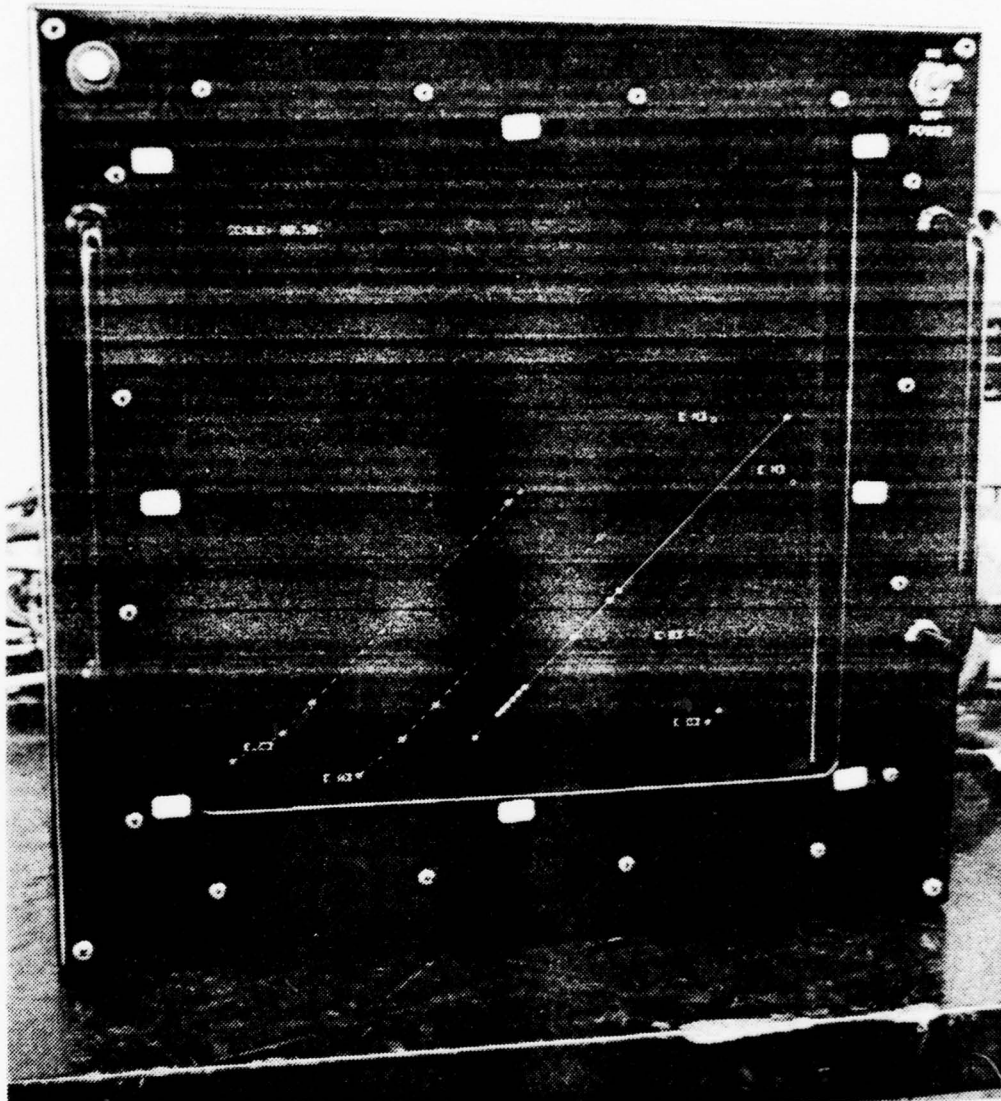


FIGURE NO. 6 PLASMA DISPLAY VIEW

by PL/M 80 [Ref. 9], which basically allows the programmer by using one identifier, to refer to a collection of STRUCTURE MEMBERS which may have different types, such as ARRAY, ADDRESS variables and BYTE variables. In PL/M 80, a BYTE variable is an 8-bit value occupying a single byte of storage, an ADDRESS variable is a 16-bit value occupying two consecutive bytes of storage, and an ARRAY is a vector composed of BYTE or ADDRESS variables. The most sophisticated data structure that PL/M 80 supports directly is the Array of Structures with Arrays inside the structures. With the capability that PL/M 80 has of allowing a variable reference to be either fully qualified, partially qualified, unqualified (references to entire arrays or structures), or by using pointers and indirect references, it was possible to simulate other more complex data structures such as circular linked lists and plexes, as will be explained later.

1. Data Structure for the System Parameters

Figure No. 7 describes the data structure used to represent most of the system parameters; as can be observed, it is a structure composed of eight (8) members with the following description:

- a. LAT: array of 4 bytes used to represent the floating point value of the latitude parameter used to define the Coordinate Grid Origin.

- b. LONG: array of 4 bytes used to represent the floating point value of the longitude parameter used to

SYSTEM

LAT (4)
LONG (4)
SCALE (4)
WIND\$DIR (4)
WIND\$SPD (4)
NUM\$ZONE (5)
CONTACT\$KIND (3)
NUMCTS (1)

FIGURE NO. 7 DATA STRUCTURE FOR THE SYSTEM PARAMETERS

define the Coordinate Grid Origin.

c. SCALE: array of 4 bytes used to represent the floating point value of the graphics scale parameter.

d. WIND\$DIR: array of 4 bytes used to represent the floating point value of the wind direction.

e. WIND\$SPD: array of 4 bytes used to represent the floating point value of the wind speed.

f. NUM\$ZONE: array of 5 bytes used to represent the ASCII characters defining the value of the Time Zone number.

g. CONTACT\$KIND: array of 3 bytes used to represent the total number of contacts in each class.

h. NUMCTS: byte variable used to represent the number of contacts at any time in the system.

It should be noticed that this data structure is designed only to maintain the parameters described at any moment; a log of changes or modifications to the parameters described is not kept.

2. Data Structures for Own Ship Parameters

Figure No. 8 describes the data structures used to represent the own ship parameters; the OWN\$SHIP\$INFO structure is used to maintain a set of parameters for which no log of changes or modifications is maintained, and also to allow indirect access to the OWN\$SHIP data structure. Its four (4) members have the following description:

a. LAT: array of 4 bytes used to represent the

[illegible]

LAT (4)
LONG (4)
POINTER (1)
FLAG (1)

51

floating point value of the latitude parameter defining the geographical position of the own ship.

b. LONG: array of 4 bytes used to represent the floating point value of the longitude parameter defining the geographical position of the own ship.

c. POINTER: byte variable used to access the OWN\$SHIP data structure, by defining its most recently used member.

d. FLAG: byte variable used to indicate whether or not the 30 members of the OWN\$SHIP data structure have been accessed at least once.

The OWN\$SHIP Array of Structures is used to maintain up to thirty (30) different sets of values for the own ship parameters, thus maintaining a log of the 30 (or fewer) most recent values of the own ship parameters. By using the members POINTER and FLAG of the OWN\$SHIP\$INFO structure, it is possible to use the OWN\$SHIP data structure as a circular queue, with each of its 30 members having the following description:

a. X: array of 4 bytes used to represent the floating point value of the X parameter defining the position of the own ship in the Coordinated Grid System defined.

b. Y: array of 4 bytes used to represent the floating point value of the Y parameter defining the position of the own ship in the Coordinated Grid System defined.

c. TIME: array of 3 bytes used to represent the real time clock value (hours, minutes and seconds) at which the corresponding member was current.

d. CRS: array of 4 bytes used to represent the floating point value of the course parameter.

e. SPD: array of 4 bytes used to represent the floating point value of the speed parameter.

Notice that each member of the OWN\$SHIP data structure is capable of providing enough information to locate the own ship; up to 30 locations and time may be recorded.

3. Data Structures for the Contact Parameters

Figure No. 9 describes the data structures used to represent contact parameters. The CONTACT\$INFO Array of Structures, composed of fifteen (15) members, was used to represent a set of parameters for up to 15 different contacts, and was also used to allow indirect access to the CONTACT\$POSI data structure; each member of CONTACT\$INFO represented a different contact and was used to access 15 corresponding members in the CONTACT\$POSI data structure, as will be described. Each member of the CONTACT\$INFO data structure had the following configuration:

a. DESIG: address value used to represent the decimal value of the designation used to refer to a contact.

b. TYPE: byte variable used to represent the type of the contact.

c. KIND: byte variable used to represent the class

CONTACT\$POST

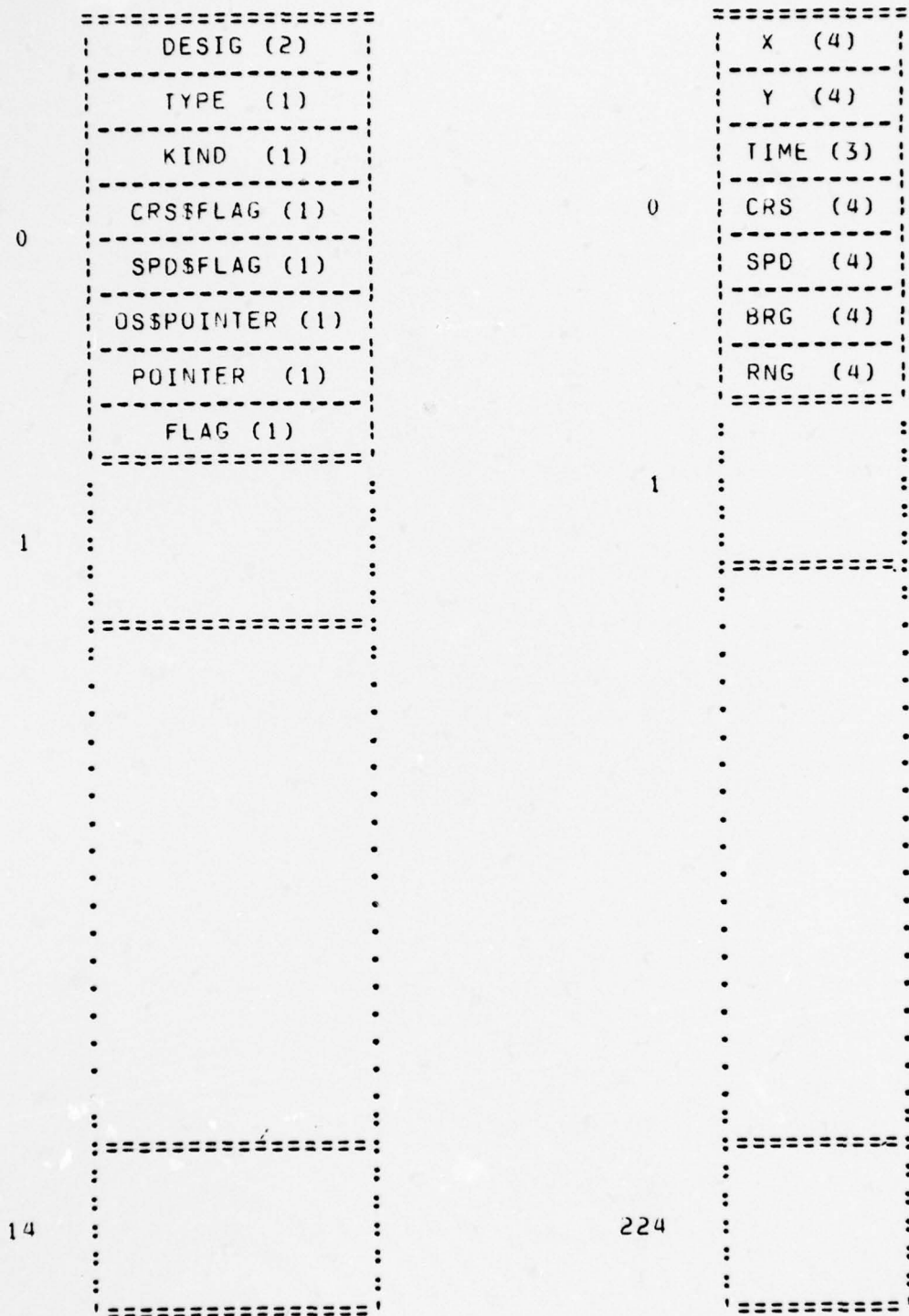


FIGURE NO. 9 DATA STRUCTURES FOR CONTACT PARAMETERS

of the contact.

d. CRS\$FLAG: byte variable used to indicate whether or not there was information about the course of the contact.

e. SPD\$FLAG: byte variable used to indicate whether or not there was information about the speed of the contact.

f. OS\$POINTER: byte variable used to indicate which of the 15 corresponding members of the CONTACT\$POSI data structure was current when the own ship made its last change in course or speed; it is mainly used when solving Maneuvering Board problems.

g. POINTER: byte variable used to access the 15 corresponding members of the CONTACT\$POSI data structure, by defining its most recently used member.

h. FLAG: byte variable used to indicate whether or not the 15 corresponding members of the CONTACT\$POSI data structure have been accessed at least once.

The CONTACT\$POSI Array of Structures is used to maintain up to 15 different sets of values for the parameters of up to 15 different contacts, thus maintaining a log of the 15 (or fewer) most recent values for the parameters of each contact. Its 225 members are accessed indirectly by the POINTER element in each member of CONTACT\$INFO, according to the following method:

Relative position of member in CONTACT\$INFO: N.

Members in CONTACT\$POSI to which N is allowed

to access: $15 \times N$ up to $(15 \times (N + 1)) - 1$

Thus, for example, the contact defined as member 0 in CONTACT\$INFO, has its 15 sets of parameters represented in members 0 thru 14 of CONTACT\$POSI, and the contact defined as member 8 in CONTACT\$INFO, has its 15 sets of parameters represented in members 120 thru 134 of CONTACT\$POSI.

Also, the 15 members of CONTACT\$POSI that are allocated for each contact defined in CONTACT\$INFO, are used as a circular queue, in a similar way to that described for the own ship.

Each of the 225 members has the following configuration:

a. X: array of 4 bytes used to represent the floating point value of the X parameter defining the position of the contact in the Coordinated Grid System defined.

b. Y: array of 4 bytes used to represent the floating point value of the Y parameter defining the position of the contact in the Coordinated Grid System defined.

c. TIME: array of 3 bytes used to represent the real time clock value (hours, minutes and seconds) at which the corresponding member was current.

d. CRS: array of 4 bytes used to represent the floating point value of the course parameter.

e. SPD: array of 4 bytes used to represent the

floating point value of the speed parameter.

f. BRG: array of 4 bytes used to represent the floating point value of the bearing parameter.

g. RNG: array of 4 bytes used to represent the floating point value of the range parameter.

VI CONCLUSION

A. RESULTS

Presently, the system will handle all functions currently provided by the DRT plotter, while at the same time solving some Maneuvering Board problems: CPA information and determination of course and speed of contacts.

Since all the inputs to the system are done manually, an extensive control program was developed in order to make it very difficult for the user to enter erroneous data, providing up to six levels of correctness.

Information about the own ship and any one of 15 different contacts can be displayed by pressing one function-defined key for each case. Also, the geographical plot displayed at the Plasma Unit can be reoriented and modified in scale by using two other function-defined keys.

The displays provided by the system can easily be duplicated, thus allowing the possibility of having all the information displayed in remote stations where needed.

B. FUTURE WORK

The system can be a valuable tool aboard non-NTDS ships as implemented. Several extensions are possible, however, in both hardware and software.

The system can be modified in order to accept input data directly from the own ship's pitometer and gyrocompass, and

also from the radar and sonar sensors, in order to reduce the time required from the operator to enter the same data, thus improving the system's performance.

The system could also be modified in order to allow the use of Magnetic Bubble Memory as mass storage media, to maintain a log of all the necessary data to be able to reconstruct events, and to start up the system again in case of a loss of power or other failure.

Also, the capability of solving other Maneuvering Board problems such as interception, scouting and torpedo firing, could easily be added.

The system could also be implemented by using several Single Board Computers working in parallel, with each one solving one or more specific problems. Also, the remote stations could be provided with intelligent terminals with the capability of displaying the information necessary, without disturbing the computer's processing.

The results of this thesis indicate that microcomputer technology can feasibly be applied to the ideas above and to other similar problems.

A microprocessor-based system as the one described, has the potential to provide any ship with a digital surface-subsurface contact plotting capability, without the expense of NTDS size equipment. It can reduce the manning of underway watches while improving the quality of the tactical information available to the Officer of the Deck.

APPENDIX A ALGORITHM DESCRIPTION

A. THE CLOSEST POINT OF APPROACH (CPA)

1. The Basic Relative Movement Problem

K. H. Kerns and R. S. Cooper [Ref. 3] have described a way of solving Maneuvering Board problems with the aid of a microcomputer. As described in that reference, Maneuvering Board problems are divided in two basic categories.

One is the relative plot where the CPA of contacts being tracked can be calculated. The center of the plot represents the "reference" or "own" ship and any other point represents the position of a "maneuvering" ship, plotted in true bearing and range from the own ship at various times.

The other category is the vector diagram or the "triangle of courses and speeds"; this allows the operator to calculate the course and speed of any maneuvering ship (a contact) given the own ship course and speed, and relative course and speed of the contact (obtained from the relative plot). Figure No. 10 shows how the two categories are applied and how they interact one with another.

2. Other Uses of the Maneuvering Board

All Maneuvering Board problems utilize the basic relative motion problem discussed above; in addition to determining CPA information, course, and speed of a contact, the Maneuvering Board can also be used to find the required course and speed to take station on or to intercept another

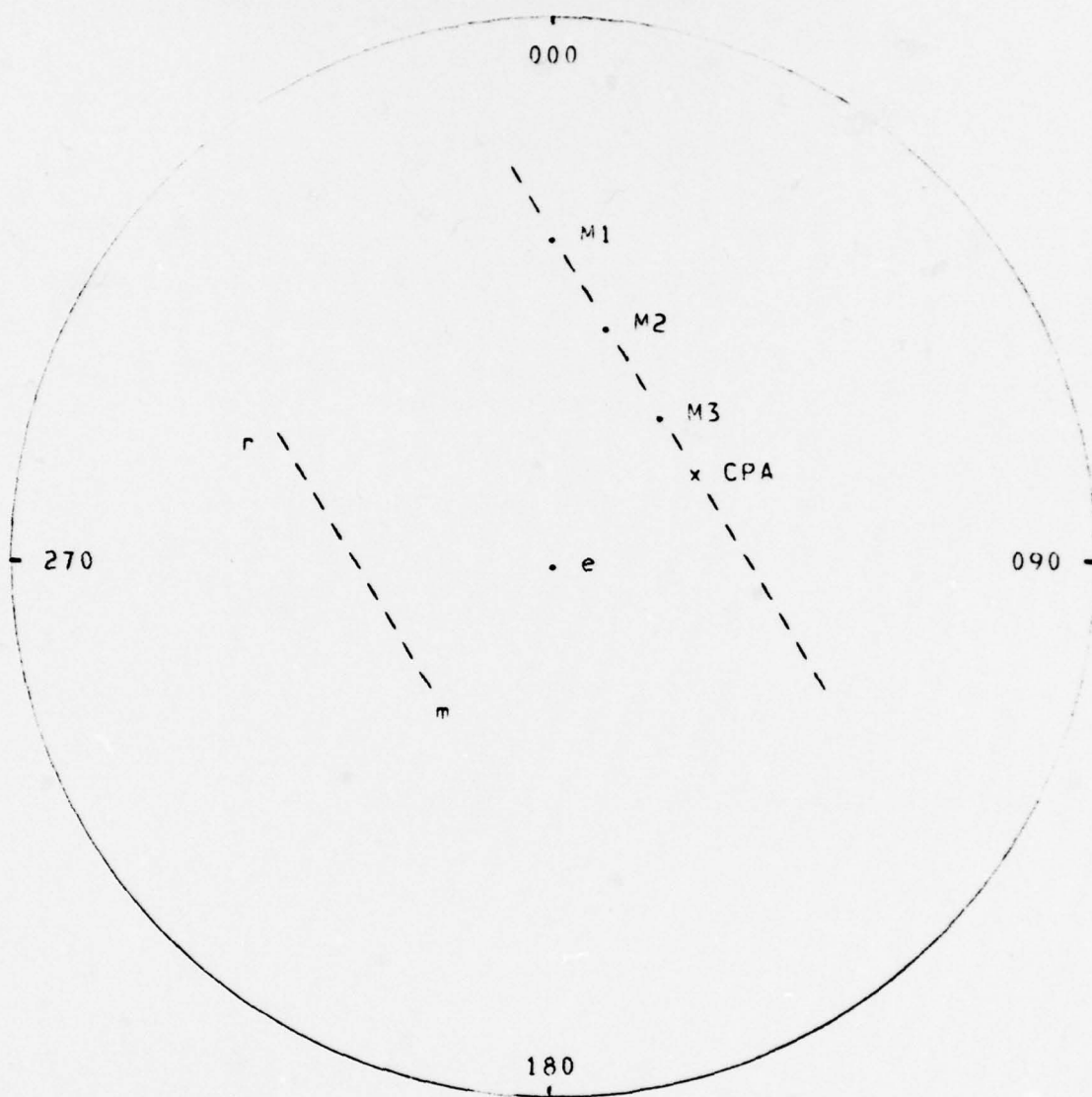


FIGURE NO. 10 MANEUVERING BOARD SCHEMATIC

ship, to find true wind, or to find courses and speeds for scouting and torpedo-firing situations.

3. Maneuvering Board problems solutions implemented

As described before, the system was mainly designed to provide a graphical display of the own ship and contacts being tracked; secondarily, the system provides a surface status display of information about those contacts and about own ship. Only the CPA, course, and speed of contacts are calculated automatically; the other functions of the Maneuvering Board are not duplicated.

4. Problems that Normally Occur

In a Maneuvering Board problem solution, all the data are recorded manually by the plotter; these data are provided aurally by the radar operator.

This interaction is somewhat error-prone; the positions plotted often appear scattered like the ones shown in Figure No. 11.

5. The Least-squares Fit Approach

In order to smooth the data utilized and to obtain a straight line representing the line of relative motion for a certain number of plotted positions, the least-square fit method was chosen.

The approach employed requires two to five contact positions. A least-square fit is used to determine the slope and the Y axis intercept. Once these parameters are obtained the CPA, course, and speed calculations are performed in a straightforward way. According to Reference No. 15 the

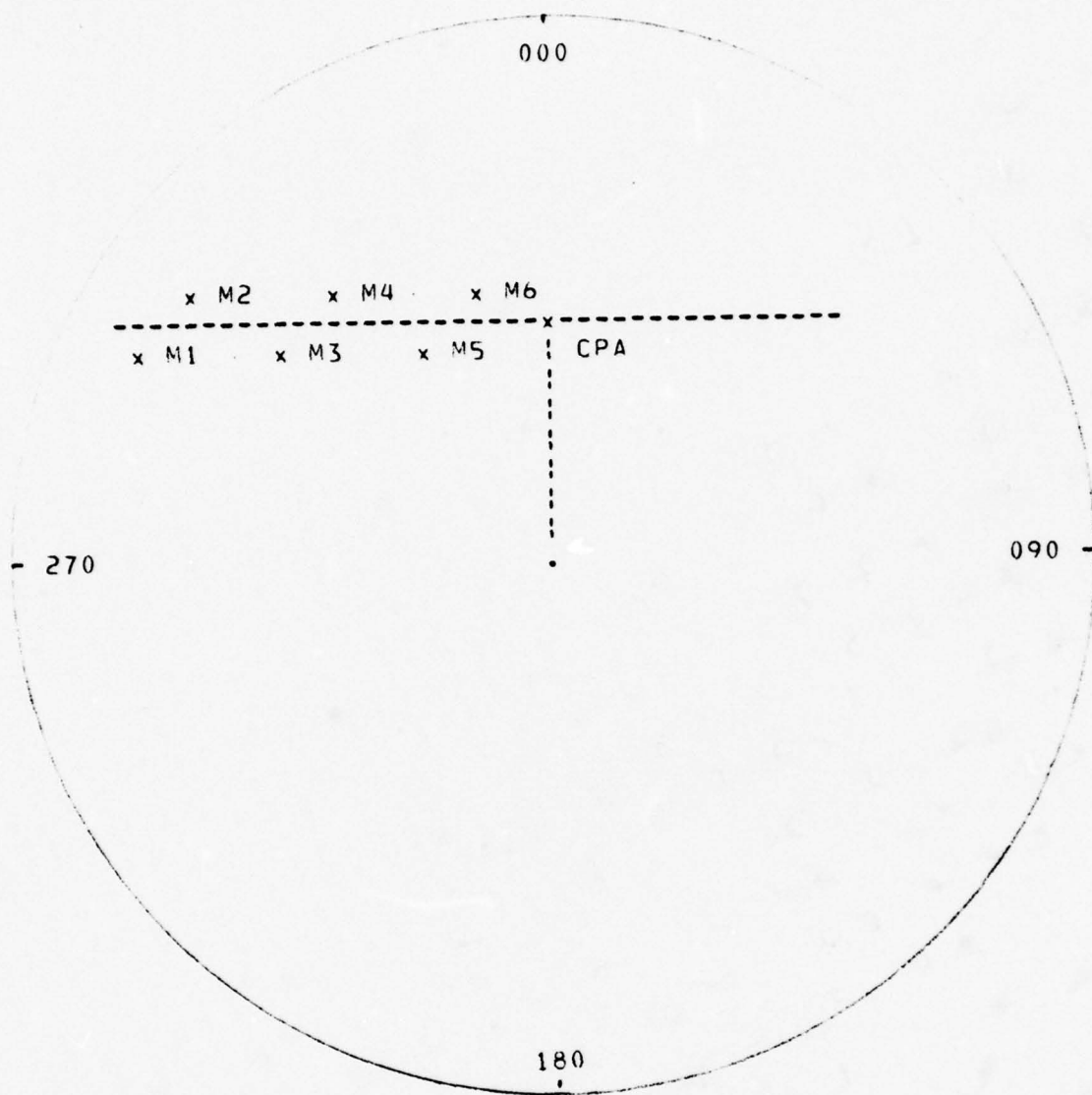


FIGURE NO. 11 USUAL PLOTTING AT THE MANEUVERING BOARD

method is as follows:

Let N be the number of positions of a contact obtained from the radar repeater/sonar (the value of N in the system has as bounds, 2 and 5). The equation of a straight line can be represented by:

$$y = MX + B$$

where:

M = slope, and

B = Y intercept.

The least-square fit method gives the solution for M and B as follows:

$$M = (s(0) \times t(1) - s(1) \times t(0)) / (s(0) \times s(2) - s(1)^2)$$

$$B = (s(2) \times t(0) - s(1) \times t(1)) / (s(0) \times s(2) - s(1)^2)$$

where:

$$s(k) = \sum_{i=0}^{N-1} X(i)^k, \quad k = 0, 1, 2$$

$$t(k) = \sum_{i=0}^{N-1} Y(i) \times X(i)^k, \quad k = 0, 1$$

6. CPA Algorithm

Certain combinations of data require special treatment in CPA, course, and speed calculations. These special cases are shown in Figure No. 12.

For case 0 the contact has the same course and speed as the own ship; then the CPA can not be calculated, because the contact is permanently at CPA.

For case 1 the contact has as direction of relative motion (relative course) the values of 000 or 180 degrees and thus the slope of the relative motion line will have an infinite value.

For case 2 the contact has as direction of relative motion the values of 090 or 270 degrees and thus the slope of the relative motion line will have the value of 0.

There is a fourth case where the contact is on a collision course with the own ship.

In the general case, the sequence of calculations is as follows:

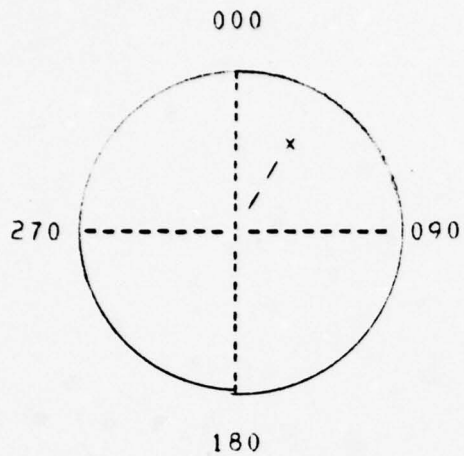
a. Take at least 2 marks of a contact (time, bearing, and distance).

b. Convert bearing and distance to relative values of x and y:

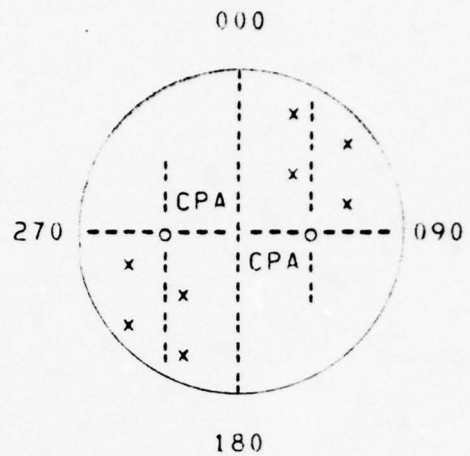
$$RELX = RNG \times \sin(BRG)$$

$$RELY = RNG \times \cos(BRG)$$

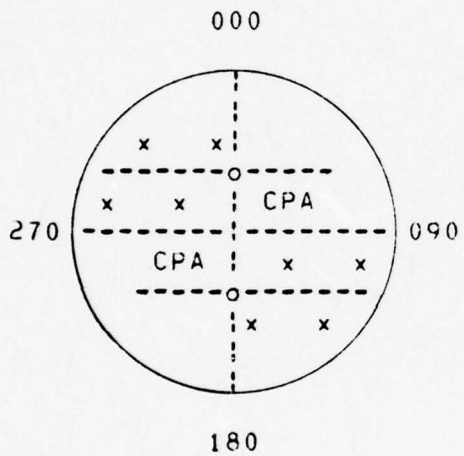
c. Compute slope of smoothed relative motion line:



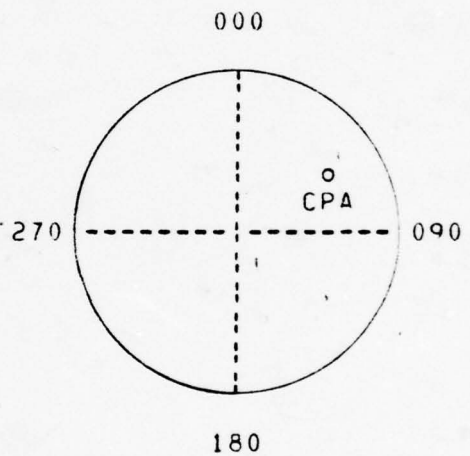
CASE 0



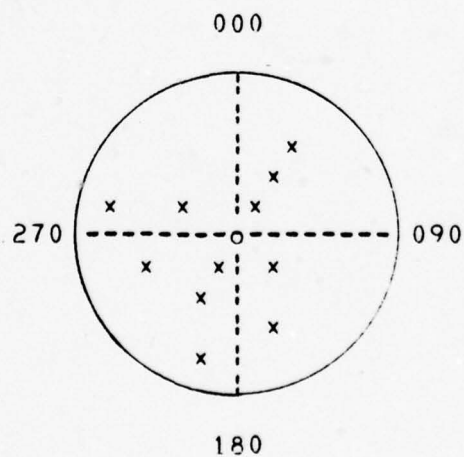
CASE 1



CASE 2



CASE 3



CASE 4

FIGURE NO. 12 SPECIAL CASES

$$M = (s(0) \times t(1) - s(1) \times t(0)) / (s(0) \times s(2) - s(1)^2)$$

d. Compute Y intercept:

$$Y\$CUT = (s(2) \times t(0) - s(1) \times t(1)) / (s(0) \times s(2) - s(1)^2)$$

e. Compute relative course:

$$Y1 = M \times REL\$X(1) + Y\$CUT$$

$$Y2 = M \times REL\$X(2) + Y\$CUT$$

$$REL\$CRS = \tan^{-1} ((REL\$X(2) - REL\$X(1)) / (Y2 - Y1))$$

where:

REL\$X(2) - obtained from the last position
used for the Least-square Fit
calculation

REL\$X(1) - obtained from the first position
used for the Least-square Fit
calculation

REL\$Y(2) - similar to REL\$X(2)

REL\$Y(1) - similar to REL\$X(1)

f. Compute relative speed:

$$DELTA\$X = REL\$X(2) - REL\$X(1)$$

$$\text{DELTA\$Y} = \text{REL\$Y}(2) - \text{REL\$Y}(1)$$

$$\text{DELTA\$T} = \text{TIME}(2) - \text{TIME}(1)$$

$$\text{REL\$SPD} = \frac{\sqrt{\text{DELTA\$X}^2 + \text{DELTA\$Y}^2}}{\text{DELTA\$T}}$$

where:

TIME(1) - local time at which the last position of a contact used in the least-square fit is entered into the system.

TIME(2) - local time at which the first position of a contact used in the least-square fit is entered into the system.

g. Compute true course and speed of a contact:

Given own ship's course (C0), own ship's speed (S0), speed of relative motion (REL\$SPD), and the relative motion course (REL\$CRS), then:

$$X1 = S0 \times \sin(C0)$$

$$Y1 = S0 \times \cos(C0)$$

$$X2 = \text{REL\$SPD} \times \sin(\text{REL\$CRS})$$

$$Y2 = \text{REL\$SPD} \times \cos(\text{REL\$CRS})$$

Thus, the X/Y components of the maneuvering ship's vector are $X1 + X2$ and $Y1 + Y2$ where the maneuvering ship's speed is:

$$\sqrt{(X1 + X2)^2 + (Y1 + Y2)^2}$$

and the course is:

$$\tan^{-1} ((X1 + X2)/(Y1 + Y2))$$

h. Compute CPA

$$X\$CPA = (M \times (M \times REL\$X(1) - Y1)) / (M^2 + 1)$$

$$Y\$CPA = (Y1 - M \times X1) / (M^2 + 1)$$

$$CPA\$TIME = \frac{\sqrt{(X\$CPA - REL\$X(1))^2 + (Y\$CPA - Y1)^2}}{REL\$SPD} + TIME(1)$$

$$CPA\$RANGE = \sqrt{X\$CPA^2 + Y\$CPA^2}$$

$$CPA\$BEARING = \tan^{-1} (X\$CPA / Y\$CPA)$$

NOTE: Y1 is the value calculated at the item e. above, not the one at g.

As a final comment, the case 4 shown in Figure No. 3 represents the situation when a contact is in collision with the own ship. It is easily seen that a contact in collision has the bearings of the various positions with approximately the same value while the range is reducing. The system was

designed with a safe CPA range value (SAFE\$RNG) as parameter which can be changed from 50 yards (default value) up to 1000 yards; thus, any CPA range below that parameter value will set the information about the CPA of a given contact as being in collision with the own ship.

Beyond that, the CPA algorithm checks for a contact that already passed its CPA and a message "MOVING AWAY" is issued.

B. GRAPHICS ON PLASMA DISPLAY

1. Physical Considerations

In order to provide elements for designing the algorithm for interacting with the plasma display some physical parameters for the AN/UYQ-10, Plasma Display set [Ref. 13] had to be taken into account:

a. Panel parameters:

Active area : 8.55" x 8.55"

Addressable matrix : 512 x 512

Dot spacing : 0.0167" center-to-center, 60 per
inch

Light spot size : 10 to 12 mils

b. Character size:

5 x 7 matrix : 80 x 120 mils

2. Plasma Display Unit Capabilities

Such capabilities included the capability to:

- a. Set status of the Plasma Unit (busy or not)
- b. Clear Plasma panel
- c. Clear vectors

- d. Receive X/Y coordinates from CPU
- e. Set alphanumeric mode
- f. Set vector mode (solid or dashed vector capability)

3. Algorithm Design

Figure No. 13 shows how the Plasma Panel was set in the coordinate system; the points 1, 3, 5, and 7 delimit the area where the Plasma Panel is located. The point 7 represents the origin of the Plasma Panel (ORIGIN\$X, ORIGIN\$Y).

In order to allow the display of all the information necessary to the Plasma Unit and generated by the system, the "PLASMA\$MODULE" module was designed.

a. Windowing

The windowing process was developed as a transformation process which enables the Plasma Panel to cover a region in the coordinate grid system.

The scale set by the operator as an initial parameter controls the windowing process and it can vary from .25 miles/inch up to 25.00 miles/inch.

In Figure No. 13 the name WINDOW marks the size of a square representing the region covered by the Plasma Panel; this value was obtained by setting:

$$\text{WINDOW} = \text{SCALE} \times 8.55$$

b. Procedure to check if a given position falls within the limits of the defined "window"

A mechanism was implemented to check if it was

possible to plot a given position (X/Y values) in the region covered by the Plasma Panel; thus, the following algorithm was developed.

In Figure No. 13 the point named P1 can be plotted in, but the point P2 can not. A point can be displayed at the Plasma Panel when the coordinates X/Y of that point follow the rules below:

$$\begin{aligned} \text{ORIGIN\$X} + \text{WINDOW} > X > \text{ORIGIN\$X}, \text{ and} \\ \text{ORIGIN\$Y} - \text{WINDOW} < Y < \text{ORIGIN\$Y} \end{aligned}$$

Notice that all those values are in floating-point representation.

c. Normalization

Before displaying a given point in the Plasma Panel it is necessary first to check if the position could be plotted, and secondly to normalize its value to the range specified (addressable matrix = 512 x 512); the first is done as explained above, and the second is as follows:

- (1) $\text{DELTA\$X} = X - \text{ORIGIN\$X}$
 $\text{DELTA\$Y} = \text{ORIGIN\$Y} - Y$
- (2) Take the absolute values of
 $\text{DELTA\$X}$ and $\text{DELTA\$Y}$:
 $\text{DELTA\$X} = \text{ABS}(\text{DELTA\$X})$
 $\text{DELTA\$Y} = \text{ABS}(\text{DELTA\$Y})$
- (3) $\text{TEMP\$X} = (511.0 / \text{WINDOW}) \times \text{DELTA\$X}$
 $\text{TEMP\$Y} = (511.0 / \text{WINDOW}) \times \text{DELTA\$Y}$
- (4) Truncate and convert to

integer representation:

$X = \text{INTEGER}(\text{TEMP\$X})$

$Y = \text{INTEGER}(\text{TEMP\$Y})$

d. Plasma Reorientation

Besides setting a scale for the "window", three ways of positioning the "window" in the coordinate grid system were implemented.

By default, every time the scale has to be changed, the last position of the own ship will be set at the center of the "window"; this is accomplished by setting:

$\text{ORIGIN\$X} = \text{OWN\$SHIP\$X} - \text{HALF\$WINDOW\$Y}, \text{ and}$

$\text{ORIGIN\$Y} = \text{OWN\$SHIP\$Y} - \text{HALF\$WINDOW}$

Notice that the own ship's last position can be set at the center of the "window" any time the operator wants to do so.

The second method implemented was to set the last known position of any contact at the center of the "window"; this was obtained by setting:

$\text{ORIGIN\$X} = \text{CONTACT\$POSIX} - \text{HALF\$WINDOW}, \text{ and}$

$\text{ORIGIN\$Y} = \text{CONTACT\$POSISY} - \text{HALF\$WINDOW}$

In case of no contact being maintained by the system, this method will be ignored by the system, even if requested.

The third method implemented was to set one of 8 fixed positions (refer to Fig. No. 13) at the center of the "window" as requested by the operator; this was obtained by setting:

(1) Point 0:

$ORIGIN\$X = ORIGIN\X , and

$ORIGIN\$Y = ORIGIN\$Y + HALF\$WINDOW$

(2) Point 1:

$ORIGIN\$X = ORIGIN\$X + HALF\$WINDOW$, and

$ORIGIN\$Y = ORIGIN\$Y + HALF\$WINDOW$

(3) Point 2:

$ORIGIN\$X = ORIGIN\$X + HALF\$WINDOW$, and

$ORIGIN\$Y = ORIGIN\Y

(4) Point 3:

$ORIGIN\$X = ORIGIN\$X + HALF\$WINDOW$, and

$ORIGIN\$Y = ORIGIN\$Y - HALF\$WINDOW$

(5) Point 4:

$ORIGIN\$X = ORIGIN\X , and

$ORIGIN\$Y = ORIGIN\$Y - HALF\$WINDOW$

(6) Point 5:

$ORIGIN\$X = ORIGIN\$X - HALF\$WINDOW$, and

$ORIGIN\$Y = ORIGIN\$Y - HALF\$WINDOW$

(7) Point 6:

$ORIGIN\$X = ORIGIN\$X - HALF\$WINDOW$, and

$ORIGIN\$Y = ORIGIN\Y

(8) Point 7:

$ORIGIN\$X = ORIGIN\$X - HALF\$WINDOW$, and

$ORIGIN\$Y = ORIGIN\$Y + HALF\$WINDOW$

C. TRANSCENDENTAL FUNCTIONS

Three transcendental functions were necessary in solving some problems by the system. These functions were sine and

cosine of a given angle, and arc tangent of the ratio of two given values.

The main goals were the minimum amount of storage for the work area and the minimum execution time in performing the calculations; for these reasons, the Hastings approximations were chosen with slight modifications made to the algorithms suggested in Ref No. 4.

1. Cosine and Sine Functions

As described in the Appendix E, the procedure "COSSSIN" performs the cosine and sine of a given angle (in radians); the following steps were taken in the development of the algorithm:

- a. Save the actual value of the angle
- b. Set angle to be between 0 and 2 x PI radians
- c. Check for special cases - 90, 270,
and 360 degrees
- d. Normalize the angle for the interval 0 and 90
degrees, and save quadrant of the
original angle
- e. Convert angle to semicircle units

$$A = \text{ANGLE} / \text{PI}$$

where $\text{PI} = 3.141593$

- f. Perform Hastings approximation

$$Z = (C1 + A^2 (C2 + A^2 (C3 + A^2 (C4 + A^2 (C5 + A^2 (A \times C6))))))A + A$$

where:

$$C1 = 0.5707963267949$$

$$C2 = -0.6459640964727$$

$$C3 = 0.0796926087138$$

$$C4 = -0.0046816668674$$

$$C5 = 0.0001602588415$$

$$C6 = -0.0000034333379$$

- a. Compute cosine and sine:

$$\cos(\text{ANGLE}) = 1.0 - 2.0 \times Z^2$$

$$\sin(\text{ANGLE}) = \sqrt{1.0 - \cos^2(\text{ANGLE})}$$

- h. Restore signs for sine and cosine according to the quadrants saved in d.

2. Arc Tangent Function

As described in Appendix E, the procedure "ARCTAN" performs the arc tangent function of a given ratio (Y/X) of 2 parameter values; the following steps were taken in the development of the algorithm:

- a. Save the actual values of the parameters
- b. Save sign of parameters to determine quadrant
- c. Check for valid arguments (X and Y)
 - (1) If X = 0 and Y = 0:
Function undefined
 - (2) If X = 0 and Y = 0:

ANGLE = 90 degrees (for Y > 0)

ANGLE = 270 degrees (for Y < 0)

d. Form Z to perform the Hastings approximation

$$Z = \frac{|Y| - |X|}{|Y| + |X|}$$

e. Perform the Hastings approximation

$$\text{ANGLE} = (C1 + Z^2 (C2 + Z^2 (C3 + Z^2 (C4 + Z^2 (C5 + Z^2 (C6 + Z^2 (C7 + Z^2 \times C8))))))Z + \text{PI}/4$$

where:

C1 = 0.9999993329

C2 = -0.3332985605

C3 = 0.1994653599

C4 = -0.1390853351

C5 = 0.0964200441

C6 = -0.0559098861

C7 = 0.0218612288

C8 = -0.0040540580

PI = 3.141593

f. Restore angle to proper quadrant

D. POSITIONAL DATA CONVERSION

In the design of the system all the positions can be referred either as latitude and longitude, or as X/Y coordinates; for this reason, some algorithms were developed in order to obtain one or another kind of positional data.

1. Convert LAT and LONG to X/Y Coordinates

The whole system was based in a Coordinate Grid System whose origin values were given in terms of latitude and longitude, and any position in it had an X/Y coordinate defined in relation to the origin; thus, given the values of latitude and longitude of a certain position, it might be converted to that Coordinate Grid System units; i.e., to convert to X/Y coordinates. This was obtained by doing:

a. Compute mean latitude:

$$\text{MEAN\$LAT} = (\text{SYSTEM\$LAT} + \text{LAT}) / 2.0$$

b. Compute X/Y coordinates:

$$X = (\text{LONG} - \text{SYSTEM\$LONG}) \times \cos(\text{MEAN\$LAT})$$

$$Y = \text{LAT} - \text{SYSTEM\$LAT}$$

2. Convert a Given Position in Terms of Bearing and Range from Own Ship to X/Y Coordinates

In order to determine the X/Y coordinates of a position when it is given in terms of bearing and range from the own ship, the following steps were done:

a. Save value of bearing:

$$\text{ANGLE} = \text{BEARING}$$

b. Compute DELTA\$X and DELTA\$Y:

$$\text{DELTA\$X} = \text{RANGE} \times \sin(\text{ANGLE})$$

$$\text{DELTA\$Y} = \text{RANGE} \times \cos(\text{ANGLE})$$

c. Compute X and Y:

$$X = \text{OWN\$SHIP\$X} + \text{DELTA\$X}$$

$$Y = \text{OWN\$SHIP\$Y} + \text{DELTA\$Y}$$

3. Convert X/Y Coordinates of a Given Position into Latitude and Longitude

a. Compute latitude:

$$\text{LAT} = Y + \text{SYSTEM\$LAT}$$

b. Compute mean latitude:

$$\text{MEAN\$LAT} = (\text{SYSTEM\$LAT} + \text{LAT}) / 2.0$$

c. Compute longitude:

$$\text{LONG} = X / \cos(\text{MEAN\$LAT}) + \text{SYSTEM\$LONG}$$

APPENDIX B SOFTWARE CATEGORIZATION

A. MODULES DESCRIPTION

As mentioned before the system was developed around 12 basic modules; there is another module (EXECUTIVE) which is embedded in the module MAIN\$MODULE.

As shown in Appendix E, every procedure has a comment header which explains what it performs, the parameters with its meaning, and, when proper, the usage of that procedure.

B. MODULES INTERACTION

Due to the interaction capability between modules as allowed by the language PL/M 80 through the use of the attributes PUBLIC and EXTERNAL for the procedures, the following list was written in order to show this interaction. This list shows the modules which have procedures called by the listed module.

1. MAIN\$MODULE:

- a. EXECUTIVE\$COMMANDS
- b. DISPLAY\$CMDS
- c. COMMANDS
- d. PLASMA\$MODULE
- e. CRT
- f. FLTASCII
- g. FLOATING\$POINT
- h. TIME
- i. BASICS

2. EXECUTIVE (embedded in MAIN\$MODULE):

- a. MAIN\$MODULE
- b. EXECUTIVE\$COMMANDS
- c. DISPLAY\$CMDS
- d. PLASMA\$MODULE
- e. CRT
- f. TIME
- g. PLASMA\$PRIMITIVES
- h. BASICS

3. EXECUTIVE\$COMMANDS:

- a. MAIN\$MODULE
- b. EXECUTIVE\$COMMANDS
- c. CPA\$MODULE
- d. COMMANDS
- e. PLASMA\$MODULE
- f. CRT
- g. FLTASCII
- h. FLOATING\$POINT
- i. TIME
- j. BASICS

4. CPA\$MODULE:

- a. EXECUTIVE\$COMMANDS
- b. CPA\$MODULE
- c. COMMANDS
- d. FLOATING\$POINT

5. DISPLAY\$CMDS:

- a. EXECUTIVE\$CMDS

- b. CPA\$MODULE
- c. DISPLAY\$CMDS
- d. COMMANDS
- e. CRT
- f. FLOATING\$POINT
- g. BASICS

6. COMMANDS:

- a. COMMANDS
- b. CRT
- c. FLTASCII
- d. FLOATING\$POINT
- e. BASICS

7. PLASMA\$MODULE:

- a. EXECUTIVE\$COMMANDS
- b. COMMANDS
- c. PLASMA\$MODULE
- d. CRT
- e. FLOATING\$POINT
- f. PLASMA\$PRIMITIVES
- g. BASICS

8. CRT:

- a. CRT
- b. BASICS

9. FLTASCII:

- a. FLTASCII
- b. FLOATING\$POINT

10. FLOATING\$POINT:

a. FLOATING\$POINT

b. BASICS

11. TIME:

a. COMMANDS

b. CRT

c. BASICS

12. PLASMA\$PRIMITIVES:

a. PLASMA\$PRIMITIVES

13. BASICS:

a. BASICS

APPENDIX C

OPERATOR'S MANUAL

for the

SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM

at the

NAVAL POSTGRADUATE SCHOOL

This manual describes the operation of the Surface-Subsurface Contact Plotter System at the Naval Postgraduate School. This manual assumes familiarization with CIC procedures. The specifics about the installation of the equipment required were presented in chapters 4 and 5 and also in Appendix D. The algorithms used were described in Appendix A. All the software required is contained in one diskette labeled PLASMA GEOGRAPHIC PLOTTER PACKAGE: SYSTEM.APL. Figure No. 14 presents a view of how the equipment is typically set up.



FIGURE NO. 14 EQUIPMENT VIEW

I. STARTUP PROCEDURES

CAUTION: NEVER turn on or off the diskette drive
with a diskette inserted !!!.

1. Turn on MDS system: use the key located at the upper left corner of the front panel and turn it clockwise. The POWER indicator should light.
2. Turn on diskette drive: use POWER switch located at the front panel. The ON indicator should light.
3. Turn on DATAMEDIA Video Terminal (CRT): use switch located on right side. The cursor should appear at the screen after a few seconds. Ensure that the lights CD, CTS, ROLL and FULL DUPLEX are on.
4. Turn on power supply to Plasma Unit. This external power supply should be set at + 5 Volts DC. A red indicator should light.
5. Turn on AN/UYO-10 Plasma Display Unit: use POWER switch located at front of unit. The indicator located at the upper left corner should light.
6. Turn on any other slave displays, if existing.
7. Place diskette labeled PLASMA GEOGRAPHIC PLOTTER PACKAGE: SYSTEM.APL in drive 0, with the read/write access slot first. Close door of the drive after diskette insertion.
8. Bootstrap the ISIS-II Operating System:
 - a. Press top of Intellec BOOT switch.

- b. Press top of RESET switch.
- c. Observe that INTERRUPT 2 indicator goes on before proceeding.
- d. Press space bar of DATAMEDIA Video Terminal keyboard.
- e. Observe that INTERRUPT 2 indicator goes off before proceeding.
- f. Press bottom of BOOT switch.
- g. Observe that the following message appears at the DATAMEDIA Video Terminal screen:

ISIS-II,V2.2

-

9. Issue the following command:

DRISYS <carriage return>

- 10. After a few seconds, the DATAMEDIA Video Terminal screen should be cleared and then filled with the working format; also, the message "ON LINE." should appear at the AN/UYO-10 Plasma Display screen.
- 11. Observe that the slave displays (if existing), are presenting the same information as their respective masters.
- 12. Follow the instructions for SYSTEM INITIALIZATION, as prompted and according to the format explained in the following pages.
- 13. Notice that the TIME value entered during SYSTEM INITIALIZATION should be that one desired as starting time (the time at which the GO key is depressed, after

the SYSTEM INITIALIZATION mode is completed).

14. During operation, the INTERRUPT 1 indicator should light (after the GO key has been depressed to start the system).

II. SHUTDOWN PROCEDURE

CAUTION: NEVER turn on or off the diskette drive
with a diskette inserted !!!.

1. Press the INTERRUPT 0 switch. The associated indicator should light.
2. Eject the diskette in drive 0.
3. Turn off the equipment in the following order:
 - a. Slave displays (if existing).
 - b. AN/UYO-10 Plasma Display Unit.
 - c. AN/UYO-10 Plasma Display Unit power supply.
 - d. DATAMEDIA Video Terminal.
 - e. Diskette drive.
 - f. Intellec MDS system.

III. FORMATS AND COMMANDS DESCRIPTION

The following pages describe the data elements, input commands, and display commands required for the operation of the SURFACE-SUBSURFACE CONTACT PLOTTER SYSTEM.

data element

data element

NAME:

Time Zone Number: parameter defining the time zone number being used to determine the local time.

FORMAT:

snn where:

s - sign (+ or -).
nn - two digit number.

RANGE:

00 <= nn <= 12

COMMENTS:

Used only for display purposes.

data element

data element

NAME:

Time: parameter defining a time value. Consists of hours, minutes and seconds.

FORMAT:

hh:mm:ss where:

hh - value of hours. Two digits.
mm - value of minutes. Two digits.
ss - value of seconds. Two digits.

RANGE:

00 <= hh <= 23
00 <= mm <= 59
00 <= ss <= 59

COMMENTS:

Once the time is set, the system will maintain the current time and update the time value displayed at the Video Terminal every second.

data element

data element

NAME:

Time Between Updates: parameter that defines the interval of time used by the system to update the geographical position of the own ship.

FORMAT:

sss where:

sss - value in seconds to be used. Three digits.

RANGE:

15 <= sss <= 250

COMMENTS:

Initially, the Time Between Updates is set automatically by the system to 180 seconds. The system will calculate the new position of the own ship and display this value at the Video Terminal and will plot, if possible, the new position at the Plasma Video any time the interval of time between the current time and the actual time is greater than or equal to the Time Between Updates parameter.

data element

data element

NAME:

Course: parameter that determines the general direction at which the own ship or any contact is steering.

FORMAT:

ddd.d where:

ddd.d - value in degrees and tenths of degrees. Three digits.

RANGE:

000.0 <= ddd.d <= 359.9

COMMENTS:

Needs to be entered for the own ship. The system will calculate its value in the case of a contact, although it could also be input for the system if known. Note that the system will eventually override this information, after solving the course problem.

AD-A059 603

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 17/7

A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOT--ETC(U)

JUN 78 A L GONCALVES, J E CUBA BRAVO

UNCLASSIFIED

NL

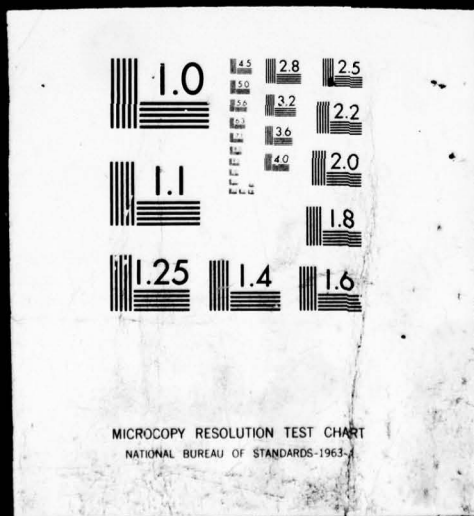
2 OF 6
AD
A059603



FILED

2 OF 6

AD
A 059603



data element

data element

NAME:

Speed: parameter that determines the velocity at which the own ship or any contact is moving.

FORMAT:

kk.k where:

kk.k - value in knots (nautical miles / hour)
and tenths of knots. Three digits.

RANGE:

00.0 <= kk.k <= 99.9

COMMENTS:

Needs to be entered for the own ship. The system will calculate its value in the case of a contact, although it could also be input for the system if known. Note that the system will eventually override this information, after solving the speed problem.

data element

data element

NAME:

Bearing: parameter that determines the true bearing of a contact from the own ship, at a given time.

FORMAT:

bbb.b where:

bbb.b - value in degrees and tenths of degrees. Four digits.

RANGE:

000.0 <= bbb.b <= 359.9

COMMENTS:

None.

data element

data element

NAME:

Range: parameter that determines the distance between the own ship and any contact, at a given time. Can be given in yards or in nautical miles.

FORMAT:

mmm.m or yyyyyy where:

mmm.m - value in miles and tenths of miles. Four digits.
yyyyyy - value in yards. Six digits.

RANGE:

000.0 <= mmm.m <= 100.0
000000 <= yyyyyy <= 999999

COMMENTS:

When the system is displaying a Range parameter, all values less than or equal to 5.0 nautical miles will be always displayed in yards. The system keeps the internal value in miles.

data element

data element

NAME:

Latitude: parameter that defines the geographical position of the own ship or any contact. Consists of a sign, degrees, minutes and tenths of minutes.

FORMAT:

dd:mm.m s where:

- dd - value in degrees. Two digits.
- mm.m - value in minutes and tenths of minutes. Three digits.
- s - sign (North (N) or South (S)).

RANGE:

00 <= dd <= 89
00.0 <= mm.m <= 59.9

COMMENTS:

The initial position of the own ship is initially required for the system; later positions of the own ship and all positions for the contacts will be automatically calculated by the system.

data element

data element

NAME:

Longitude: parameter that defines the geographical position of the own ship or any contact. Consists of a sign, degrees, minutes and tenths of minutes.

FORMAT:

ddd.mm.m s where:

ddd - value in degrees. Three digits.

mm.m - value in minutes and tenths of minutes. Three digits.

s - sign (East (E) or West (W)).

RANGE:

00 <= ddd <= 179

00.0 <= mm.m <= 59.9

COMMENTS:

The initial position of the own ship is initially required for the system; later positions of the own ship and all positions for the contacts will be automatically calculated by the system.

data element

data element

NAME:

X: parameter that defines the position of the own ship or any contact in the Coordinate Grid System being used. Its value is given in miles.

FORMAT:

snnnnnnnnnn.nn where:

s - sign (+ or -).
nnnnnnnnnn.nn - value in miles and
hundredths of miles.
Twelve digits.

RANGE:

0.0 <= !nnnnnnnnnn.nn! <= 9999999999.99

COMMENTS:

Is not determined directly by the operator; it is calculated automatically by the system which also takes care of its format and range. Could be positive or negative.

data element

data element

NAME:

Y: parameter that defines the position of the own ship or any contact in the Coordinate Grid System being used. Its value is given in miles.

FORMAT:

snnnnnnnnnn.nn where:
s - sign (+ or -).
nnnnnnnnnn.nn - value in miles and
hundredths of miles.
Twelve digits.

RANGE:

0.0 <= !nnnnnnnnnn.nn! <= 9999999999.99

COMMENTS:

Is not determined directly by the operator; it is calculated automatically by the system which also takes care of its format and range. Could be positive or negative.

data element

data element

NAME:

Designation (Desig): parameter that defines the name given to a particular contact. Consists of two alphabetic characters or one blank and one alphabetic character.

FORMAT:

Aa where:

- A - Any alphabetic character or a blank.
- a - Any alphabetic character.

RANGE:

Not applicable.

COMMENTS:

Needs to be unique for the set of contacts maintained by the system at any given time. Upper or lower case characters can be used, although the system will change all lower case characters to their upper case equivalents.

data element

data element

NAME:

Type: parameter that defines if the contact is of surface or sub-surface kind.

FORMAT:

t where:

t - could be S (Surface) or SS (Sub-surface).

RANGE:

Not applicable.

COMMENTS:

None.

data element

data element

NAME:

Class: parameter that defines the identity and purpose of any contact.

FORMAT:

c where:

c - can have three values:

F if Friendly (FRI).

H if Hostile (HOS).

U if Unknown (UNK).

RANGE:

Not applicable.

COMMENTS:

Depending on the Class of any contact, the symbol used to plot its positions at the Plasma Display will vary: a circle is used for Friendly contacts, and an 'X' is used for Hostile and Unknown contacts.

data element

data element

NAME:

Scale: parameter that defines the scale at which the picture presented at the Plasma Unit is displayed. Can be specified up to one hundredth of a mile/inch.

FORMAT:

mm.mm where:
mm.mm - value in miles and hundreths of miles per inch. Four digits.

RANGE:

00.25 <= mm.mm <= 25.00

COMMENTS:

The Scale value is used in determining the size of the window used in forming the picture to be displayed on the Plasma Unit.

data element

data element

NAME:

Safe CPA Range: parameter that defines the radius of a circle with center at the own ship; any contact that will pass through this security circle will be considered in collision.

FORMAT:

yyyy where:
yyyy - value in yards. Four digits.

RANGE:

0050 <= yyyy <= 1000

COMMENTS:

Initially the Safe CPA Range parameter is set automatically by the system to a value of 0050 yards.

data element

data element

NAME:

Wind Direction: this parameter indicates the true bearing from which the wind is blowing.

FORMAT:

ddd.d where:

ddd.d - value in degrees and tenths of degrees.
Four digits.

RANGE:

000.0 <= ddd.d <= 359.9

COMMENTS:

Used only for display purposes.

data element

data element

NAME:

Wind Speed: this parameter indicates the speed at which the wind is blowing.

FORMAT:

kk.k where:

kk.k - value in knots (nautical miles / hour)
and tenths of knots. Three digits.

RANGE:

00.0 <= kk.k <= 99.9

COMMENTS:

Used only for display purposes.

input command

input command

NAME:

Origin Update.

DESCRIPTION:

This command is used to modify the Coordinate Grid Origin parameters.

INPUT REQUIRED:

- New Latitude and Longitude values.

OPTIONAL INPUT:

- None.

COMMENTS:

This command causes the system to change all the X/Y values that had been calculated, and also to redraw the picture represented at the Plasma Display with the last position of the own ship at the center.

input command

input command

NAME:

Own Ship Update.

DESCRIPTION:

This command is used to modify the parameters of the own ship: Latitude, Longitude, Course and Speed, in a selective way.

INPUT REQUIRED:

- When prompted by the system, indicate which parameters are going to be changed.

OPTIONAL INPUT:

- New Latitude value.
- New Longitude value.
- New Course value.
- New Speed value.

DESCRIPTION:

When this command is issued and when the system prompts the user to indicate which parameters will be changed, if the user responds that no parameter is desired to be changed, then the control will be passed to the system again.

The time information needed will be automatically obtained by the system, and will be the time at which this command was issued.

If any parameter is changed, this causes the system to automatically update the position of the own ship to the moment at which the command was issued.

All changes made will be displayed at the Video Terminal and reflected in the Plasma Display.

If the Latitude and/or Longitude values are changed, then all X/Y values in the system will be recalculated and a new picture with the new position of the own ship at the center will be presented at the Plasma Display.

Notice also that when the Course and/or Speed parameters are changed, no CPA information will be available for any contact until a new mark is obtained for any contact from

which CPA information is desired.

input command

input command

NAME:

Create.

DESCRIPTION:

This command is issued to record a new contact.

INPUT REQUIRED:

- Designation.
- Type.
- Class.
- Bearing.
- Range.

OPTIONAL INPUT:

- Course and Speed values if known.

COMMENTS:

When this command is issued, the position of the own ship is updated to that moment; the first position of the new contact will be displayed at the Plasma Unit, if possible (if it is inside the window).

The system will check for the Designation given to the new contact, to make sure that it is not being utilized at that moment; if such an error is detected, a warning message will be displayed.

The time information needed for the contact will be automatically obtained by the system, and will be the time at which this command was issued.

Notice that when the system already has 15 contacts, this command will not be accepted, and a warning message will be issued.

If the system is maintaining less than six contacts, the information entered for the new contact will be automatically displayed at the Surface Status Board presented in the Video Terminal. In the case of six or more existing contacts, no information will be automatically displayed.

input command

input command

NAME:

Remove Contact.

DESCRIPTION:

This command is used to remove a contact from the system.

INPUT REQUIRED:

- Assurance that this command is really desired.

OPTIONAL INPUT:

- Designation of the contact desired to be removed.
- Designation of a contact desired to be displayed.

COMMENTS:

The system will prompt the user to determine if this command is really needed; if the user made a mistake in calling this command this is the moment to rectify it. If the user agrees that this command is really needed, then the system will prompt for the designation of the contact desired to be deleted.

If the designation given to the system does not exist, the system will issue a warning message, and will again try to determine if this command is really desired.

If the contact removed was being displayed at the Surface Status Board, its information will be erased; if at this moment all the contacts in the system are being displayed at the Surface Status Board, then the control will pass again to the system; however, if not all the contacts were being displayed, the system will prompt for the Designation of a contact that the user desires to be displayed in place of the recently removed contact; the system at this point will check that the Designation given really exists and that it does not correspond to a contact already being displayed; if any error is discovered, a warning message will be issued.

If the contact just removed was not being displayed at the Surface Status Board, the control will pass automatically to the system.

If the contact just removed was being displayed at the Plasma Unit, its position(s) will not be erased until a new

picture is drawn.

The information about the class of the contacts in the system will be also automatically updated.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

input command

input command

NAME:

Redesignate.

DESCRIPTION:

This command is used to give a new Designation to any contact already in the system.

INPUT REQUIRED:

- Old Designation.
- New Designation.

OPTIONAL INPUT:

- None.

COMMENTS:

This command will check for both Designations to be correct. If any one of them represents an error - the Old Designation does not exist or the New Designation already exists - a warning message will be issued. The Old Designation will not be accepted as New Designation at the same time.

If the contact being redesignated is displayed at the Surface Status Board, its DESIG field will be automatically updated.

If the contact being redesignated is displayed at the Plasma Unit, its DESIG field will not be automatically updated until the next time the picture is drawn.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

input command

input command

NAME:

Contact Update.

DESCRIPTION:

This command is used to update the information about any contact being maintained by the system: Type, Class, Bearing, Range, Course and Speed, may be changed selectively.

INPUT REQUIRED:

- Designation of the contact desired to be updated.
- Indication of which parameters are desired to be updated.

OPTIONAL INPUT:

- New Type value.
- New Class value.
- New Bearing value.
- New Range value.
- New Course value.
- New Speed value.

COMMENTS:

If the user responds that no parameter is desired to be changed, the control will pass automatically to the system.

The system will prompt for only those parameters that were indicated by the user.

The time information needed for the contact will be automatically obtained by the system, and will be the time at which this command was issued.

If the Bearing and/or Range parameters are changed, this will cause the system to automatically update the position of the own ship to the moment at which this command was issued.

The system will check that the Designation given is correct, and if an error is detected, then a warning message will be issued.

If the contact being updated is also displayed at the Surface Status Board, its new parameters will be displayed

after the command finishes.

The new position determined will be displayed at the Plasma Unit if possible.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

input command

input command

NAME:

Swap Contacts.

DESCRIPTION:

This command is used to change the list of contacts that are being displayed on the Surface Status Board.

INPUT REQUIRED:

- Designation of a contact desired to be out of the display.
- Designation of a contact desired to be in the display.

OPTIONAL INPUT:

- None.

COMMENTS:

The system will check for the following possible errors:

- A non-existent Designation is given.
- The Designation of a contact that is not at the display is given to indicate the contact desired to be out of the display.
- The Designation of a contact already in the display is given to indicate the new contact desired to be in the display.

In any of these cases, the system will issue a warning message and continue to prompt.

This command will not be accepted if the number of contacts in the system is less than seven, and a warning message will be issued.

input command

input command

NAME:

Time.

DESCRIPTION:

This command is used to update/change all the parameters that the system has with respect to time: Time Zone Number, System Clock Value and Time Between Updates.

INPUT REQUIRED:

- Indication of which parameters are desired to be updated.

OPTIONAL INPUT:

- New Time Zone Number value.
- New System Clock value.
- New Time Between Updates.

COMMENTS:

If the user responds that no parameter is desired to be changed, the control will pass automatically to the system.

If the Time Zone Number is modified, the new value will be automatically displayed at the Video Terminal.

If the Time Between Updates is modified and the new value is smaller than the old one, the elapsed time from the last automatic update of the own ship's position will be compared against the new Time Between Updates, and if appropriate, the position of the own ship will be updated.

input command

input command

NAME:

CPA Safe Range Update.

DESCRIPTION:

This command is used to update/change the value of the CPA Safe Range parameter.

Remember that the initial default value of the CPA Safe Range is 0050 yards.

INPUT REQUIRED:

- New CPA Safe Range value.

OPTIONAL INPUT:

- None.

COMMENTS:

The new CPA Safe Range will be used in all following CPA calculations, and the Display will not be affected to reflect this change until such new calculations occur.

input command

input command

NAME:

Wind Update.

DESCRIPTION:

This command is used to introduce/update information about the wind.

INPUT REQUIRED:

- New Wind direction value.
- New Wind speed value.

OPTIONAL INPUT:

- None.

COMMENTS:

This information is used for display purposes only.

input command

input command

NAME:

Scale Update.

DESCRIPTION:

This command is used to modify the value of the Scale parameter being used to define the window that limits the picture to be represented at the Plasma Display.

INPUT REQUIRED:

- New Scale value.

OPTIONAL INPUT:

- None.

COMMENTS:

This command will cause the system to define a new window to be used in forming the picture to be represented at the Plasma Display; a new picture will be presented reflecting the change made, with the last position of the own ship at the center.

Notice that the value of the Scale parameter is permanently displayed at the Plasma Unit.

input command

input command

NAME:

Plasma Reorient.

DESCRIPTION:

This command is used to redefine the position of the window used to form the picture to be presented at the Plasma Display.

INPUT REQUIRED:

- Type of reorientation desired.

OPTIONAL INPUT:

- Value of new point to be at the center of the new window.
- Designation of a contact whose last position is desired to be at the center of the new window.

COMMENTS:

This command allows the user to redefine the position of the window in three different ways:

- By selecting one of eight predefined points to be the center of the new window.
- By making the last position of the own ship to be the center of the new window.
- By selecting the last position of any contact to be the center of the new window. This possibility is allowed only when there is at least one contact in the system. The system will also check for a valid Designation, and will issue a warning message if necessary.

This command will cause the system to draw a new picture at the Plasma Display, according to the picture defined by the new position of the window.

display command

display command

NAME:

Origin.

DESCRIPTION:

This command is used to display the values of the Coordinate Grid Origin: Latitude and Longitude.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Coordinate Grid Origin Latitude value.
- Coordinate Grid Origin Longitude value.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

display command

display command

NAME:

Scale.

DESCRIPTION:

This command is used to display the value of the Scale parameter currently in use.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Scale parameter value.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

Notice that the value of the Scale parameter is permanently displayed at the Plasma Unit.

display command

display command

NAME:

Own Ship.

DESCRIPTION:

This command is used to display the parameters associated with the last position of the own ship.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- First page: Positional data.
 - 1) Latitude and Longitude values.
 - 2) X and Y values of the last determined position.
- Second page: Tactical data.
 - 1) Time of last Course and/or Speed change.
 - 2) Course and Speed in effect at the moment.

COMMENTS:

This information is displayed using the lower portion of the screen.

The different pages are obtained by pressing the 'GO' key each time a new page is desired.

The control will return to the system after the last page has been passed, by pressing the 'GO' key.

display command

display command

NAME:

Contact Information.

DESCRIPTION:

This command is used to display information about any contact being maintained by the system.

INPUT REQUIRED:

- Designation of a contact whose information is desired.

INFORMATION DISPLAYED:

- First page: General data.
 - 1) Designation, Type, Class, and Current Number of Positions in the system.
- Second page: Positional data.
 - 1) Latitude and Longitude values of the last determined position.
 - 2) X and Y values of the last determined position.
- Third page: Tactical data.
 - 1) Time at which the present data was obtained.
 - 2) Bearing and Range values of last mark.
 - 3) Course and Speed values if available.
- Fourth page: CPA data.
 - 1) Time at which CPA will occur, if applicable.
 - 2) Bearing and Range defining the CPA, if applicable.
 - 3) Note: if CPA can not be calculated a message will be displayed in place of this information. Also, if a special CPA case is present, one of the following messages will be displayed:
 - "COLLISION AT ..(time).." (Blinking).
 - "MOVING AWAY".
 - "SAME COURSE AND SPEED".
- Fifth page: Actual Estimated Position.
 - 1) Bearing and Range defining the estimated position of the contact. This will only be displayed if the Course and Speed values of the contact are known.

COMMENTS:

This information is displayed using the lower portion of the screen.

The different pages are obtained by pressing the 'GO' key each time a new page is desired.

The control will return to the system after the last page has been passed, by pressing the 'GO' key.

The system will check for a valid Designation, and a warning message will be issued if a mistake is detected.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

display command

display command

NAME:

Contacts in System.

DESCRIPTION:

This command is used to obtain information about the Designation of all the contacts in the system.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Designations of the contacts maintained by the system.

COMMENTS:

This information is displayed by using the lower portion of the screen.

This command will not be accepted if there are no contacts in the system, and a warning message will be issued.

display command

display command

NAME:

Request CPA Safe Range.

DESCRIPTION:

This command is used to obtain information about the current value of the CPA Safe Range.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Actual CPA Safe Range value.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

display command

display command

NAME:

Wind.

DESCRIPTION:

This command is used to obtain information about the wind.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Wind direction and Wind speed values if available. If there is no information about the wind, a warning message will be issued.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

display command

display command

NAME:

Time Between Updates.

DESCRIPTION:

This command is used to obtain information about the current value of the Time Between Updates parameter.

INPUT REQUIRED:

- None.

INFORMATION DISPLAYED:

- Actual value of the Time Between Updates parameter.

COMMENTS:

This information is displayed using the lower portion of the screen.

The control will pass to the system after pressing the 'GO' key when indicated.

Notice that the default value of this parameter is 180 seconds.

APPENDIX D FLOATING-POINT HARDWARE BOARD

A. GENERAL INFORMATION

The floating-point package developed for this system is based on the SBC 310 High-speed Mathematics Unit from INTEL Corporation. As described by the Reference No. 10, the SBC 310 unit is a member of a complete line of the INTEL SBC 80 system expansion modules. In performing high-speed mathematic functions, the Math Unit acts as an intelligent processor slaved to one or more SBC 80 computer masters. The Math Unit performs its repertoire of 14 arithmetic functions an order of magnitude faster than is possible with software routines.

B. DESCRIPTION OF THE MATH UNIT

The Math Unit is a microprogrammed processor on a single board and is designed to be plugged into a standard SBC 604/614 Modular Backplane and Cardcage to interface directly with an SBC 80 Single Board Computer or to be used with an INTEL Intellec Microcomputer Development System (MDS).

Standard operations include floating-point add, subtract, multiply, divide, square, and square root; fixed-point integer multiply, divide, and extended divide; conversions between fixed and floating-point representations; and test, compare, and argument exchange operations.

The Math Unit implements unbiased rounding for maximum

accuracy. Unbiased rounding is the same as ordinary rounding unless the result is exactly midway between two floating-point numbers; in this case, ordinary rounding always increases the result, whereas unbiased rounding rounds the result to the nearest even number. When a calculation is performed that results in either an exponent underflow or overflow, the Math Unit provides exponent wraparound to prevent loss of information.

Operation codes for invoking the arithmetic functions are passed to the Math Unit via I/O Write commands, which are also used to initialize the unit with a memory base address. I/O Read commands are used to determine the Math Unit Status. Arguments are passed to the Math Unit via Memory Write commands and the results are obtained via Memory Read commands.

The Math Unit, which can be operated either in the Interrupt or Polled mode, generates a Busy signal during processing operations and generates either a Complete signal or an Error signal after the computation is complete. The information to the host computer which these three signals convey is explained in Reference No. 10.

The memory base address and I/O base address are user selectable. The 16-bit memory address is completely under software control and is assigned by the host processor through a sequence of I/O Write commands addressed to the Math Unit. The 8-bit I/O base address is selected by a dual inline package (DIP) switch on the board.

All Math Unit operations, including arithmetic calculations, data flow between functional elements on the board bus interface, and associated logical tasks, are resident microprogram permanently stored in a set of eight INTEL 3604 Erasable Programmable Read Only Memory (EPROM) chips. This memory provides 1,024 micro-instructions of 32 bits each.

C. PREPARATION FOR USE

1. Installation Considerations

The Math Unit board is designed for interface with an INTEL SBC 80 Single Board Computer based system or an INTEL Intellec Microcomputer Development System (MDS).

When installing the SBC 310 in an INTEL Intellec MDS, the CPU board needs to be reconfigured in order to generate a Qualified Write Signal; this reconfiguration was obtained by overriding the advanced acknowledge (AACK) feature, by moving a jumper labeled "advanced write" from a D-C connection to an E-D connection and by disabling the AACK/line (pin 25) on the CPU board as shown in the schematic diagram at page 3-47 of the Reference No. 6.

Other details about installation can be seen on Chapter 2 of the Reference No. 10.

2. I/O Base Address Switches

The host processor transmits control information and receives status information from the Math Unit by issuing I/O Write and I/O Read commands, respectively. The I/O address used for these commands is relative to an 8-bit base

address that must be a multiple of 8. This base address is assigned by the user by means of an 8-pole dual inline package (DIP) switch assembly. Five of the eight switch poles are connected to the I/O base address detection logic; the other three poles are unused.

The Math Unit used had its DIP switch set to the I/O base address of 10 hexadecimal (only switch pole no. 4 was set on).

3. Programming Information

The I/O base address, which must be assigned by switch selection before the memory base address can be assigned, is normally performed as part of the initial installation procedure. This switch setting allows the user to establish a reference or base to the ports being used in the I/O operations; Table No. IV shows the configuration of the I/O addressing as it was set in the system.

The memory base address, which is software controlled, is assigned by a sequence of two I/O write commands. The first command is addressed to port P+1 and loads the low-order byte of the memory base address. The second command is addressed to port P+2 and loads the high-order byte of the memory base address. The memory base address must be a multiple of 16; i.e., the lower byte must be in the form X0H (X is any hexadecimal digit) to accommodate the 16 required memory locations used in the arithmetic operations. In the system developed for this thesis the memory base address was set at 0F790H. After both

I/O PORT ADDRESS	OUTPUT	INPUT
P = 010H	OP CODE	R
011H	MEM LOW	STATUS BYTE
012H	MEM HIGH	R
013H	R	R
014H	R	R
015H	R	R
016H	R	R
017H	R	FLAG BYTE
P: I/O base address. R: Reserved. OP CODE: Mathematic function; see Table I. MEM LOW: Memory base address (lower byte). MEM HIGH: Memory base address (upper byte).		

TABLE IV. I/O ADDRESSING

bytes are output, the memory base address (M) is established and need not be reloaded during any subsequent operations. An initialization routine for establishing the memory base address was designed and it can be seen in the "INIT\$FP" procedure in the floating-point module (see "FLOATING\$POINT" module at Appendix E).

4. Math Unit Functions

The Math Unit performs floating-point arithmetic, fixed-point integer arithmetic, compare and test operations, and float-to-fix and fix-to-float conversions. Operation codes and execution times for the various functions are listed in Table No. I. Arithmetic and conversion formats are shown in Table No. V.

Beyond the functions performed by the Math Unit, the Floating-point package was designed with two more procedures which compute the cosine and sine of a given angle and the arc tangent value of the ratio of two given arguments.

It was observed that the compare operation between two floating-point numbers did not work properly when the two numbers were both negative; due to this fact, further code was implemented in the "FCMPR" procedure as can be seen in the Appendix E.

5. Argument and Result Data Formats

Argument and result data formats and memory locations for the various operations are presented in Table No. VI. For each argument and result, this table includes a FORMAT number cross-referenced to one of the four formats

FORMAT NO.	SINGLE PRECISION FLOATING POINT
1	<div> <pre> s7 1022 16 15 8 7 0 E EE F F F F F F '---v---'---v---'---v---'---v---' M+3 M+2 M+1 M </pre> </div> <p>where: M = memory base address. S = "0" = positive; "1" = negative. E7-E0 = biased exponent; bias = 07FH. F22-F0 = fraction; F is always normalized.</p>
2	<div> <pre> 15 8 7 0 F F F F '---v---'---v---' M+1 M </pre> </div> <p>where: M = memory base address. F15-F0 = 16-bit integer (unsigned).</p>
2A	<div> <pre> 31 24 23 16 15 8 7 0 F F F F F F F F F '---v---'---v---'---v---'---v---' M+3 M+2 M+1 M </pre> </div> <p>where: M = memory base address. F31-F0 = 32-bit integer (unsigned).</p>
3	<div> <pre> s30 24 23 16 15 8 7 0 F F F F F F F F F '---v---'---v---'---v---'---v---' M+3 M+2 M+1 M </pre> </div> <p>where: M = memory base address. S = "0" = positive; "1" = negative. F30-F0 = two's complement integer.</p>
	CONVERSION FUNCTIONS

TABLE V. ARITHMETIC AND CONVERSION FORMATS

shown in Table No. V. Table No. VI also includes the OP CODE for each operation. It is important to note that the result of an operation replaces the first argument in memory, and that the second argument may be destroyed in the course of the computation; these side effects were avoided in the floating-point software design by saving the original values and by allowing the user the possibility of having one of the operands as the result. Error conditions for each operation are described in the next paragraph.

6. Status and Flags

The Math Unit may be operated in the interrupt mode or polled mode.

In the interrupt mode, the Math Unit may be wire-wrapped to initiate an interrupt request under one or both of the following conditions:

- a. Operation complete without an error.
- b. Operation complete with an error.

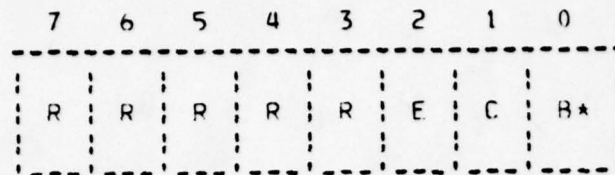
These "completion" signals may be individually wire-wrapped to separate interrupt lines or both "completion" signals may be wire-wrapped to the same interrupt line [Ref. No. 10].

In the polled mode, the subroutine designed for this purpose checks both the status byte and the flag byte (Tables No. VII and No. VIII). The polled mode procedure loops on testing the busy bit until the busy bit is clear, and then checks the error bits. If an error exists, the error code is input from the Math Unit and a message error

OP	OP CODE	ARGUMENT FORMAT*	ARGUMENTS**	RESULT FORMAT*	RESULT***
MUL	0	2	M, M+1 M+4, M+5	2A	M, M+1, M+2, M+3
DIV	1	1	M, M+1 M+4, M+5	2	M, M+1 (rem. in M+4, M+5)
EDIV	E	2A 2	M, M+1, M+2, M+3 M+4, M+5	2A	M, M+1, M+2, M+3 (rem. in M+4, M+5, M+6, M+7)
FMUL	2	1	M, M+1, M+2, M+3	1	M, M+1, M+2, M+3
FDIV	3		M+4, M+5, M+6, M+7		
FADD	4				
FSUB	5				
FSQR	6	1	M, M+1, M+2, M+3	1	M, M+1, M+2, M+3
FSQRT	7				
FLTDS	8	3	M, M+1, M+2, M+3	1	M, M+1, M+2, M+3
FIXSD	9	1	M, M+1, M+2, M+3	3	M, M+1, M+2, M+3
FCMPR	A	1	M, M+1, M+2, M+3	-	STATUS byte
FZTST	B	1	M, M+1, M+2, M+3	-	STATUS byte
EXCH	F	Any	M, M+1, M+2, M+3 M+4, M+5, M+6, M+7	-	Rotates both arguments.

* Refer to appropriate FORMAT NO column in Table V.
 ** Second argument is always the operator, and may be destroyed during the operation.
 *** Results of all operations, except FIXSD are rounded. FIXSD truncates the result.

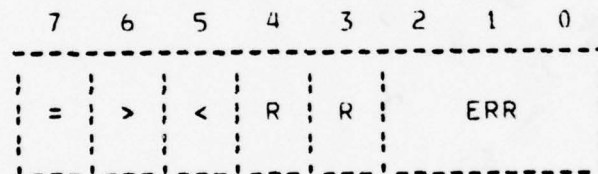
TABLE VI. OPERATION ARGUMENT AND RESULT DATA FORMATS



where: R is reserved for future use.
 B is busy.
 C is operation complete without error.
 E is operation complete with error.

* When B = 1, the Math Unit is busy and cannot respond to further requests except requests for flags.

TABLE VII. FLAG BYTE FORMAT



where: R is reserved for future use.
 = is equal (for FCMPR and FZTST).
 > is greater than (for FCMPR and FZTST).
 < is less than (for FCMPR and FZTST).

ERR is a 3-bit error code specifying one of the following error conditions:

000	No error.
001	Divide by zero.
010	Domain error.
011	Overflow.
100	Underflow.
101	First argument invalid.
110	Second argument invalid.
111	Reserved.

TABLE VIII. STATUS BYTE FORMAT

is issued. The software developed for control of the Math Unit in the system employs the polled mode described.

As mentioned before, the condition of the Math Unit is continuously updated and stored. The flag byte shown in Table No. VII may be obtained by performing an I/O Read command to P+7 (Table No. IV). After an operation is completed, the status byte may be obtained by performing an I/O Read command to P+1 (Table IV).

As shown in Table No. VIII, the status byte indicates error conditions where applicable and the results of Compare ("FCMPR" procedure) and Test ("FZIST" procedure) operations. Each of the six error conditions are defined as follows:

a. Divide by Zero (001) - This error condition is returned by either "DIV", "EDIV", or "FDIV" procedures to indicate that an attempt was made to divide by zero.

b. Domain Error (010) - This error condition is returned by the "FSQRT" procedure to indicate that the argument was not in the domain of the function; i.e., an attempt was made to take the square root of a negative number.

c. Overflow (011) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", "FSQR", and "FIXSD" procedures. In the case of "FIXSD" procedure, this error indicates that the floating-point number is too large to be converted to a 32-bit two's complement signed integer. If an overflow error occurs during "FIXSD", the floating-

point argument is left unchanged and may be read from the Math Unit.

In all other cases, this error condition signifies that the exponent of the result is too large to be represented in eight bits. In this case, 0BEH is subtracted from the resulting exponent (bringing it back into range for other computations and ensuring a valid result), and the lower eight bits of the exponent are returned in the exponent field of the result.

d. Underflow (100) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", and "FSQR" procedures to indicate that the exponent of the result is too small to be represented in eight bits. In this case, 0BEH is added to the resulting exponent (bringing it back into range for other computations and ensuring a valid result), and the lower eight bits of the exponent are returned in the exponent field of the result.

e. First Argument Invalid (101) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV", "FSQR", "FSQRT", "FIXSD", "FCMPR", and "FZTST" procedures to indicate that the first (or only) argument for the specified function is invalid. The second argument (if applicable) is not checked if this error is encountered. The invalid argument is left unchanged and may be read from the Math Unit.

f. Second Argument Invalid (110) - This error condition is returned by the "FADD", "FSUB", "FMUL", "FDIV",

and "FCMPR" procedures to indicate that the second argument for the specified function is invalid. This error condition occurs only after the first argument is checked and found valid. The invalid argument is left unchanged and may be read from the Math Unit.

NOTES:

- (1) The floating-point argument may be expressed as:

$$(-1) \times (2^{E - \text{bias}}) \times (1.F)$$

Notice that a "1" is assumed in the highest position of F.

- (2) There is one unique representation for zero:

$$S = 0$$

$$E7-E0 = 0$$

$$F22-F0 = 0$$

- (3) The following representations are invalid:

- (a) assures unique representation of zero

$$E7-E0 = 0, \text{ and}$$

$$S \neq 0 \text{ or } F22-F0 \neq 0$$

- (b) reserved for future enhancements

$$E7-E0 = 0FFH$$

7. Examples of Floating-point Number Representation

As shown in Table No. V the representation of a floating-point number has one particularity that needs to be mentioned; i.e., a "1" is always assumed in the highest bit position, and then it yields an effective 24-bit mantissa. For the sake of clarity some examples are given in Table No. IX.

F.P. NUMBER	M	M+1	M+2	M+3
-5.0	00H	00H	0A0H	0C0H
-3.0	00H	00H	40H	0C0H
-2.0	00H	00H	00H	0C0H
-1.0	00H	00H	80H	0BFH
0.0	00H	00H	00H	00H
+1.0	00H	00H	80H	3FH
+2.0	00H	00H	00H	40H
+3.0	00H	00H	40H	40H
+5.0	00H	00H	0A0H	40H
+5.4	0CDH	0CCH	0ACH	40H

TABLE IX. FLOATING-POINT NUMBERS

APPENDIX E PROGRAM LISTINGS

A. EXTERNAL DECLARATIONS

1. EXTER.....	157
2. EXTER\$1.....	167
3. EXTER\$2.....	172

B. PROCEDURES BY MODULE

1. MAIN\$MODULE.....	174
NO\$WIND.....	179
NO\$CONTACT.....	180
NOT\$ENOUGH\$CONTACTS.....	181
TOO\$MANY\$CONTACTS.....	182
MOVE\$OWN\$SHIP.....	183
EXECUTIVE (not a procedure).....	186
2. EXECUTIVE\$COMMANDS.....	193
DE\$HASH.....	200
CHECK\$GO\$KEY.....	201
DISPLAY\$KIND.....	202
CHECK\$DESIG.....	203
CONV\$MIN\$RAD.....	204
CONV\$RAD\$MIN.....	205
CONV\$XY.....	206
CONV\$REL\$XY.....	208
INIT\$STRUCTURES.....	210
GET\$SYSTEM\$PARAMETERS.....	212
DISPLAY\$CONTACT.....	216

CREATE.....	218
REMOVE.....	222
REDESIGNATE.....	226
UPDATE.....	229
SWAP\$CONTACTS.....	235
TRANSLATE.....	238
OWN\$SHIP\$UPDATE.....	240
ORIGIN.....	245
WIND.....	246
SCALE.....	248
GET\$SAFE\$RNG.....	249
INPUT\$TIME.....	251
3. CPA\$MODULE.....	254
CONV\$CONTACT\$TIME.....	255
CPA\$TIME\$CONV.....	257
CONTACT\$CRS\$SPD.....	259
CPA\$CALCULATION.....	261
GET\$CPA.....	274
4. DISPLAY\$CMDS.....	277
CONV\$LAT\$LONG.....	279
DISPLAY\$DESIG.....	281
DISPLAY\$TYPE.....	282
DISPLAY\$CLASS.....	283
DISPLAY\$LAT\$LONG.....	284
DISPLAY\$XY.....	286
DISPLAY\$CRS\$BRG.....	287
DISPLAY\$SPD.....	288

DISPLAY\$RANGE.....	289
DISPLAY\$TIME.....	290
DISPLAY\$ORIGIN.....	292
DISPLAY\$SCALE.....	293
DISPLAY\$OWN\$SHIP.....	294
DISPLAY\$CONTACT\$INFO.....	296
DISPLAY\$SYSTEM.....	304
DISPLAY\$SAFE\$RNG.....	306
DISPLAY\$WIND.....	307
DISPLAY\$UPDATE\$TIME.....	309
5. COMMANDS.....	310
PRINT\$ERROR\$MSG.....	314
CHECK\$YES\$NO.....	315
CHECK\$FP\$VALUE.....	316
CHECK\$INPUT.....	317
GET\$DEGREES.....	318
GET\$MINUTES.....	320
GET\$SIGN.....	321
FP\$FORMAT.....	322
RANGE\$FORMAT.....	325
LAT\$LONG\$FORMAT.....	327
GET\$TIME\$ZONE.....	329
GET\$LAT.....	331
GET\$LONG.....	333
GET\$COURSE\$BRG.....	335
GET\$SPEED.....	337
GET\$RANGE.....	339

GET\$DESIG.....	342
GET\$TYPE.....	344
GET\$KIND.....	346
GET\$SCALE.....	348
6. PLASMA\$MODULE.....	350
SET\$WINDOW.....	353
CLEAR\$STRUCTURES.....	354
DRAW\$FRIENDLY\$SYMBOL.....	355
DRAW\$UNK\$HOS\$SYMBOL.....	357
DRAW\$OWN\$SHIP\$SYMBOL.....	358
CHECK\$PLASMA.....	359
NORMALIZE.....	361
PUT\$OS\$CENTER.....	363
PUT\$CONTACT\$CENTER.....	364
FIXED\$REORIENTATION.....	365
PLASMA\$REDESIG.....	367
PLASMA\$DELETE.....	368
PLASMA\$CONTACT.....	369
PLASMA\$OS.....	371
DRAW\$EVERYTHING.....	373
DISPLAY\$PLASMA\$SCALE.....	375
REORIENT\$PS.....	376
7. CRT.....	380
CRT\$MASTER\$CLEAR.....	382
SET\$LOW\$HOME.....	383
CLEAR\$LOW\$SCREEN.....	384
SET\$HIGH\$HOME.....	385

PUT\$SPACE.....	386
PUT\$TAB.....	387
PUT\$FS.....	388
PUT\$LF.....	389
START\$PROT\$FIELD.....	390
START\$BLINK.....	391
STOP\$PROT\$FIELD.....	392
INTERP.....	393
INIT\$HIGH\$SCREEN.....	395
PRINT\$TIME\$ZONE.....	397
PRINT\$TIME.....	398
PRINT\$LAT\$LONG.....	399
PRINT\$COURSE.....	400
PRINT\$SPEED.....	401
PRINT\$CONTACTS.....	402
PRINT\$MODE.....	403
PRINT\$CONTACT\$INFO.....	404
8. FLTASCII.....	405
ASCII\$TO\$FLOAT.....	408
FRAC\$TO\$ASCII.....	411
FLOAT\$TO\$ASCII.....	413
9. FLOATING\$POINT.....	417
INIT\$FP.....	420
ADJUST\$OP.....	421
ADJUST1\$OP.....	422
ADJUST2\$OP.....	423
VAL\$RESULT.....	424

VAL\$RESULT\$1.....	425
VAL\$RESULT\$2.....	426
COMPARE.....	427
FLOAT\$MSG\$ERROR.....	429
CHECK.....	431
MUL.....	432
DIV.....	433
EDIV.....	434
FMUL.....	435
FDIV.....	436
FADD.....	437
FSUB.....	438
FSQR.....	439
FLTDS.....	440
FIXSD.....	441
FSQRT.....	442
FCMPR.....	443
FZTST.....	445
EXCH.....	447
COS\$SIN.....	448
ARC\$TAN.....	453
10. TIME.....	457
CLOCK.....	459
INITIATE\$TIME.....	461
INITIATE\$CLOCK.....	464
ACTUAL\$TIME.....	465
11. PLASMA\$PRIMITIVES.....	466

SET\$STATUS\$PLASMA.....	468
PLASMA\$WRITE.....	469
CLEAR\$PLASMA.....	470
PLASMA\$WRITE\$VECTOR.....	471
PLASMA\$PRINT\$STRING.....	472
INITIALIZE\$PLASMA.....	473
SET\$VECTOR.....	474
START\$VECTOR\$SOLID.....	475
STOP\$VECTOR\$SOLID.....	476
START\$VECTOR\$DASH.....	477
STOP\$VECTOR\$DASH.....	478
GRAPHIC\$DESIG.....	479
12. BASICS.....	481
CRT\$WRITE.....	482
CRT\$PRINT\$STRING.....	483
CRT\$READ.....	484
CRT\$TRY\$READ.....	485
ECHO\$CRT.....	486
SEND\$SUB.....	487
SEND\$CR.....	488
SEND\$LF.....	489
SEND\$CRLF.....	490
SEND\$BEL.....	491
SEND\$BS.....	492
SEND\$SPACE.....	493
BYTE\$CHAR.....	494
ADDRESS\$CHAR.....	495

BYTE\$TO\$ASCII.....	496
GET\$BYTÉ.....	497
GET\$ADDRESS.....	499
GET\$STRING.....	501
PUT\$NUMBER\$BUFFER.....	503

C. LISTINGS:

EXTER

EXTER

```
/*** EXTER: ***/  
  
DECLARE  
  TIME$BUFFER(6) BYTE EXTERNAL,  
  RES$TABLE(8) BYTE EXTERNAL,  
  <MILI$SEC,DUMMY$SEC,SECONDS,MINUTES,HOURS,DAY,SEC$TIME> BYTE EXTERNAL,  
  TIME$STEP ADDRESS EXTERNAL,  
  MEBA$ ADDRESS EXTERNAL;  
  
CRT$WRITE:  
  PROCEDURE (CHAR) EXTERNAL;  
  DECLARE CHAR BYTE; END;  
  
CRT$PRINT$STRING:  
  PROCEDURE (A) EXTERNAL;  
  DECLARE A ADDRESS; END;  
  
CRT$READ:  
  PROCEDURE BYTE EXTERNAL;  
  END;  
  
CRT$TRY$READ:  
  PROCEDURE BYTE EXTERNAL;  
  END;  
  
ECHO$CRT:  
  PROCEDURE BYTE EXTERNAL;  
  END;  
  
SEND$SUB:  
  PROCEDURE EXTERNAL;
```

EXTER

EXTER

END;

SEND\$CR:
PROCEDURE EXTERNAL;
END;

SEND\$LF:
PROCEDURE EXTERNAL;
END;

SEND\$CRLF:
PROCEDURE EXTERNAL;
END;

SEND\$BEL:
PROCEDURE EXTERNAL;
END;

SEND\$BS:
PROCEDURE EXTERNAL;
END;

SEND\$SPACE:
PROCEDURE (NUM) EXTERNAL;
DECLARE NUM BYTE; END;

BYTE\$CHAR:
PROCEDURE (CHAR) EXTERNAL;
DECLARE CHAR BYTE; END;

ADDRESS\$CHAR:

ENTER

ENTER

```
PROCEDURE (CHAR) EXTERNAL;  
DECLARE CHAR ADDRESS; END;  
  
BYTE$TO$ASCII:  
PROCEDURE (A,B,C) EXTERNAL;  
DECLARE (A,B,C) ADDRESS; END;  
  
GET$BYTE:  
PROCEDURE (A) BYTE EXTERNAL;  
DECLARE A BYTE; END;  
  
GET$ADDRESS:  
PROCEDURE (A) ADDRESS EXTERNAL;  
DECLARE A BYTE; END;  
  
GET$STRING:  
PROCEDURE (A,B) EXTERNAL;  
DECLARE A ADDRESS, B BYTE; END;  
  
PUT$NUMBER$BUFFER:  
PROCEDURE (A,B) EXTERNAL;  
DECLARE A BYTE, B ADDRESS; END;  
  
INIT$FP:  
PROCEDURE EXTERNAL;  
END;  
  
MUL:  
PROCEDURE (A,B,C) EXTERNAL;  
DECLARE (A,B,C) ADDRESS; END;
```

EXTER

EXTER

DIV: PROCEDURE (A,B,C,D) EXTERNAL;
 DECLARE (A,B,C,D) ADDRESS; END;

EDIV: PROCEDURE (A,B,C,D) EXTERNAL;
 DECLARE (A,B,C,D) ADDRESS; END;

FMUL: PROCEDURE (A,B,C) EXTERNAL;
 DECLARE (A,B,C) ADDRESS; END;

FDIV: PROCEDURE (A,B,C) EXTERNAL;
 DECLARE (A,B,C) ADDRESS; END;

FADD: PROCEDURE (A,B,C) EXTERNAL;
 DECLARE (A,B,C) ADDRESS; END;

FSUB: PROCEDURE (A,B,C) EXTERNAL;
 DECLARE (A,B,C) ADDRESS; END;

FSQR: PROCEDURE (A,B) EXTERNAL;
 DECLARE (A,B) ADDRESS; END;

FSQRT: PROCEDURE (A,B) EXTERNAL;
 DECLARE (A,B) ADDRESS; END;

EXTER

EXTER

```
FLTDS:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

FIXSD:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

FCMPR:
  PROCEDURE (A,B,C) BYTE EXTERNAL;
  DECLARE (A,B,C) ADDRESS; END;

FZTST:
  PROCEDURE (A,B) BYTE EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

EXCH:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

COS$SIN:
  PROCEDURE (A,B,C) EXTERNAL;
  DECLARE (A,B,C) ADDRESS; END;

ARC$TAN:
  PROCEDURE (A,B,C) EXTERNAL;
  DECLARE (A,B,C) ADDRESS; END;

ASCII$TO$FLOAT:
  PROCEDURE (A,N,B) EXTERNAL;
```

EXTER

EXTER

DECLARE (A,B) ADDRESS, N BYTE; END;

Float\$to\$ASCII:

PROCEDURE (A,B,C) EXTERNAL;

DECLARE (A,B,C) ADDRESS; END;

INITIATE\$TIME:

PROCEDURE EXTERNAL;

END;

INITIATE\$CLOCK:

PROCEDURE EXTERNAL;

END;

ACTUAL\$TIME:

PROCEDURE EXTERNAL;

END;

CRT\$MASTER\$CLEAR:

PROCEDURE EXTERNAL;

END;

SET\$LOW\$HOME:

PROCEDURE EXTERNAL;

END;

CLEAR\$LOW\$SCREEN:

PROCEDURE EXTERNAL;

END;

SET\$HIGH\$HOME:

ENTER

ENTER

```
PROCEDURE EXTERNAL;  
END;
```

```
INIT$HIGH$SCREEN;  
PROCEDURE EXTERNAL;  
END;
```

```
START$BLINK;  
PROCEDURE EXTERNAL;  
END;
```

```
PRINT$TIME$ZONE;  
PROCEDURE (A) EXTERNAL;  
DECLARE A ADDRESS; END;
```

```
PRINT$TIME;  
PROCEDURE (A) EXTERNAL;  
DECLARE A ADDRESS; END;
```

```
PRINT$LAT$LONG;  
PROCEDURE (A,B) EXTERNAL;  
DECLARE (A,B) ADDRESS; END;
```

```
PRINT$COURSE;  
PROCEDURE (A) EXTERNAL;  
DECLARE A ADDRESS; END;
```

```
PRINT$SPEED;  
PROCEDURE (A) EXTERNAL;  
DECLARE A ADDRESS; END;
```

ENTER

ENTER

```
PRINT$CONTACTS:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;

PRINT$MODE:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;

PRINT$CONTACT$INFO:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE A BYTE, B ADDRESS; END;

CHECK$YES$NO:
  PROCEDURE BYTE EXTERNAL;
  END;

CHECK$FP$VALUE:
  PROCEDURE (A,B) BYTE EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

CHECK$INPUT:
  PROCEDURE BYTE EXTERNAL;
  END;

GET$DEGREES:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE A BYTE, B ADDRESS; END;

GET$MINUTES:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;
```


EXTER

EXTER

```
GET$SIGN:
  PROCEDURE (A,B) BYTE EXTERNAL;
  DECLARE (A,B) BYTE; END;

FP$FORMAT:
  PROCEDURE (A,B,C,D) BYTE EXTERNAL;
  DECLARE (A,B) ADDRESS, (C,D) BYTE; END;

RANGE$FORMAT:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

LAT$LONG$FORMAT:
  PROCEDURE (A,B,C) EXTERNAL;
  DECLARE (A,B) ADDRESS, C BYTE; END;

GET$TIME$ZONE:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;

GET$LAT:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;

GET$LONG:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;

GET$COURSE$BRG:
  PROCEDURE (A,B) EXTERNAL;
```

EXTER

EXTER

DECLARE A BYTE, B ADDRESS; END;

GET\$SPEED:
PROCEDURE (A) EXTERNAL;
DECLARE A ADDRESS; END;

GET\$RANGE:
PROCEDURE (A) EXTERNAL;
DECLARE A ADDRESS; END;

GET\$DESIG:
PROCEDURE ADDRESS EXTERNAL;
END;

GET\$TYPE:
PROCEDURE BYTE EXTERNAL;
END;

GET\$KIND:
PROCEDURE BYTE EXTERNAL;
END;

GET\$SCALE:
PROCEDURE (A) EXTERNAL;
DECLARE A ADDRESS; END;

EXTER\$ONE

EXTER\$ONE

```
/**/ EXTER$ONE:  /**/

DECLARE LIT LITERALLY 'LITERALLY',
      DCL LIT 'DECLARE',

DCL SYSTEM STRUCTURE
  (LAT (4) BYTE,
   LONG (4) BYTE,
   SCALE (4) BYTE,
   WIND$DIR (4) BYTE,
   WIND$SPD (4) BYTE,
   NUM$ZONE (5) BYTE,
   CONTACT$KIND (3) BYTE,
   NUMCTS BYTE ) EXTERNAL,

OWN$SHIP$INFO STRUCTURE
  (LAT (4) BYTE,
   LONG (4) BYTE,
   POINTER BYTE,
   FLAG BYTE) EXTERNAL,

OWN$SHIP (30) STRUCTURE
  (X (4) BYTE,
   Y (4) BYTE,
   TIME (3) BYTE,
   CRS (4) BYTE,
   SPD (4) BYTE) EXTERNAL,

CONTACT$INFO (15) STRUCTURE
  (DESIG ADDRESS,
   TYPE BYTE,
```

EXTER\$ONE

EXTER\$ONE

KIND BYTE,
CRS\$FLAG BYTE,
SPD\$FLAG BYTE,
OS\$POINTER BYTE,
POINTER BYTE,
FLAG BYTE> EXTERNAL,

CONTACT\$POSI (225) STRUCTURE
<X (4) BYTE,
Y (4) BYTE,
TIME (3) BYTE,
CRS (4) BYTE,
SPD (4) BYTE,
BRG (4) BYTE,
RNG (4) BYTE> EXTERNAL;

DCL CONTACT\$DISPLAY (6) BYTE EXTERNAL;

DCL
LAT\$STRING (9) BYTE EXTERNAL,
LONG\$STRING (9) BYTE EXTERNAL,
CRS\$STRING (6) BYTE EXTERNAL,
SPD\$STRING (5) BYTE EXTERNAL,
CONTACTS\$STRING (8) BYTE EXTERNAL,
CONTACT\$INFO\$STRING (44) BYTE EXTERNAL;

DE\$HASH:
PROCEDURE (A,B) EXTERNAL;
DCL (A,B) ADDRESS; END;

EXTER\$ONE

EXTER\$ONE

```
CHECK$GO$KEY:
  PROCEDURE EXTERNAL;
END;

CONV$MIN$RAD:
  PROCEDURE (A,B) EXTERNAL;
  DCL (A,B) ADDRESS; END;

DISPLAY$KIND:
  PROCEDURE EXTERNAL;
END;

CHECK$DESIG:
  PROCEDURE (A) BYTE EXTERNAL;
  DCL A ADDRESS; END;

CONV$RAD$MIN:
  PROCEDURE (A,B) EXTERNAL;
  DCL (A,B) ADDRESS; END;

CONV$XY:
  PROCEDURE (A,B,C,D) EXTERNAL;
  DCL (A,B,C,D) ADDRESS; END;

CONV$REL$XY:
  PROCEDURE (A,B,C,D) EXTERNAL;
  DCL (A,B,C,D) ADDRESS; END;

GET$SYSTEM$PARAMETERS:
  PROCEDURE EXTERNAL;
```

EXTER\$ONE

EXTER\$ONE

END;

DISPLAY\$CONTACT:

PROCEDURE (A,B) EXTERNAL;
DCL (A,B) BYTE; END;

CREATE:

PROCEDURE EXTERNAL;
END;

REMOVE:

PROCEDURE EXTERNAL;
END;

REDESIGNATE:

PROCEDURE EXTERNAL;
END;

UPDATE:

PROCEDURE EXTERNAL;
END;

SWAP\$CONTACTS:

PROCEDURE EXTERNAL;
END;

OWN\$SHIP\$UPDATE:

PROCEDURE EXTERNAL;
END;

ORIGIN:

EXTER\$ONE

EXTER\$ONE

PROCEDURE EXTERNAL;
END;

WIND:
PROCEDURE EXTERNAL;
END;

SCALE:
PROCEDURE EXTERNAL;
END;

INPUT\$TIME:
PROCEDURE BYTE EXTERNAL;
END;

EXTER\$TWO

EXTER\$TWO

```
/**/  EXTER$TWO:  /**/

SET$STATUS$PLASMA:
  PROCEDURE (A) EXTERNAL;
  DECLARE A BYTE; END;

PLASMA$WRITE:
  PROCEDURE (A) EXTERNAL;
  DECLARE A BYTE; END;

CLEAR$PLASMA:
  PROCEDURE EXTERNAL;
  END;

PLASMA$WRITE$VECTOR:
  PROCEDURE (A) EXTERNAL;
  DECLARE A ADDRESS; END;

PLASMA$PRINT$STRING:
  PROCEDURE (A, B, C) EXTERNAL;
  DECLARE (A, B) BYTE, C ADDRESS; END;

INITIALIZE$PLASMA:
  PROCEDURE EXTERNAL;
  END;

SET$VECTOR:
  PROCEDURE (A, B, C) EXTERNAL;
  DECLARE (A, B, C) ADDRESS; END;
```


EXTER\$TWO

EXTER\$TWO

START\$VECTOR\$SOLID:
 PROCEDURE (A, B) EXTERNAL;
 DECLARE (A, B) ADDRESS; END;

STOP\$VECTOR\$SOLID:
 PROCEDURE (A, B) EXTERNAL;
 DECLARE (A, B) ADDRESS; END;

START\$VECTOR\$DASH:
 PROCEDURE (A, B) EXTERNAL;
 DECLARE (A, B) ADDRESS; END;

STOP\$VECTOR\$DASH:
 PROCEDURE (A, B) EXTERNAL;
 DECLARE (A, B) ADDRESS; END;

GRAPHIC\$DESIG:
 PROCEDURE (A, B, C) EXTERNAL;
 DECLARE (A, B, C) ADDRESS; END;

MAIN\$MODULE

MAIN\$MODULE

MAIN\$MODULE: DO;

MAIN\$MODULE

EXTERNALS

```
/*** EXTERNALS: ***/  
  
$NOLIST  
$INCLUDE (:F1:EXTER.SRC)  
$INCLUDE (:F1:EXTER1.SRC)  
$LIST  
  
CONV$LAT$LONG:  
  PROCEDURE (A,B,C,D) EXTERNAL;  
  DCL (A,B,C,D) ADDRESS; END;  
  
DISPLAY$WIND:  
  PROCEDURE EXTERNAL;  
  END;  
  
DISPLAY$CONTACT$INFO:  
  PROCEDURE EXTERNAL;  
  END;  
  
DISPLAY$ORIGIN:  
  PROCEDURE EXTERNAL;  
  END;  
  
DISPLAY$SCALE:  
  PROCEDURE EXTERNAL;  
  END;
```

MAIN\$MODULE

EXTERNALS

```
DISPLAY$OWN$SHIP:  
  PROCEDURE EXTERNAL;  
END;
```

```
DISPLAY$SAFE$RNG:  
  PROCEDURE EXTERNAL;  
END;
```

```
DISPLAY$SYSTEM:  
  PROCEDURE EXTERNAL;  
END;
```

```
DISPLAY$UPDATE$TIME:  
  PROCEDURE (A) EXTERNAL;  
  DECLARE A BYTE; END;
```

```
GET$SAFE$RNG:  
  PROCEDURE EXTERNAL;  
END;
```

```
CLEAR$PLASMA:  
  PROCEDURE EXTERNAL;  
END;
```

```
INITIALIZE$PLASMA:  
  PROCEDURE EXTERNAL;  
END;
```

```
SET$WINDOW:
```


MAIN\$MODULE

```
PROCEDURE EXTERNAL;  
END;
```

```
CLEAR$STRUCTURES;  
PROCEDURE EXTERNAL;  
END;
```

```
PUT$OS$CENTER;  
PROCEDURE EXTERNAL;  
END;
```

```
PLASMA$OS;  
PROCEDURE EXTERNAL;  
END;
```

```
DRAW$EVERYTHING;  
PROCEDURE EXTERNAL;  
END;
```

```
DISPLAY$PLASMA$SCALE;  
PROCEDURE EXTERNAL;  
END;
```

```
REORIENT$PS;  
PROCEDURE EXTERNAL;  
END;
```

EXTERNALS

MAIN\$MODULE

DECLARATIONS

```

/+++  DECLARATIONS:  ***/

DCL TRUE LIT '0FFH',
FALSE LIT '00H',
FOREVER LIT 'WHILE TRUE',
PROMPT LIT '025H',
DISPLAY$UPPER$LIMIT LIT '01AH',
INPUT$LOWER$LIMIT LIT '02DH',
INPUT$UPPER$LIMIT LIT '03AH';

DCL DISPLAY (*) BYTE DATA ('DISPLAY$$'),
INPUT  (*) BYTE DATA (' INPUT $$');

DCL TIME$LIMIT BYTE,
TEMP BYTE,
WIND$FLAG BYTE,
COMMAND BYTE;

```

MAIN\$MODULE

NO\$WIND

```

/*****
* NO$WIND:
* THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT NO WIND INFORMATION
* EXISTS IN THE SYSTEM.
*
*****/
NO$WIND: PROCEDURE;
  DCL MSG (*) BYTE DATA
    ('NO WIND INFORMATION AVAILABLE. $$');

  CALL CRT$PRINT$STRING(, MSG);
  CALL SEND$CRLF;
  CALL SEND$CRLF;
  CALL CHECK$GO$KEY;
  CALL CLEAR$LOW$SCREEN;
  END NO$WIND;

```

MAIN\$MODULE

NO\$CONTACT

```

/*****
*
* NO$CONTACT:
* THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE SYSTEM IS NOT CUR-
* RENTLY MAINTAINING ANY CONTACT.
*
*****/
NO$CONTACT: PROCEDURE;
    DCL MSG (*) BYTE DATA
        ('NO CONTACTS ARE BEING MAINTAINED BY THE SYSTEM. $$');

    CALL CRT$PRINT$STRING(, MSG);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
    END NO$CONTACT;

```


MAIN\$MODULE

NOT\$ENOUGH\$CONTACTS

```

/*****
*
* NOT$ENOUGH$CONTACTS:
* THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE NUMBER OF CONTACTS
* PRESENTLY AT THE SYSTEM, IS NOT ENOUGH TO ALLOW SWAPING OF CONTACTS NOT
* IN THE SCREEN.
*
*****/
NOT$ENOUGH$CONTACTS: PROCEDURE;
DCL MSG (*) BYTE DATA
      ('ALL CONTACTS IN THE SYSTEM ARE ALREADY DISPLAYED. $$');

CALL CRT$PRINT$STRING(.MSG);
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
END NOT$ENOUGH$CONTACTS;

```

MAIN\$MODULE

TOO\$MANY\$CONTACTS

```
/*  
* TOO$MANY$CONTACTS:  
* THIS PROCEDURE IS USED TO TELL THE OPERATOR THAT THE SYSTEM CAN NOT ACCEPT  
* ANY MORE NEW CONTACTS.  
*  
*****  
TOO$MANY$CONTACTS: PROCEDURE;  
  DCL MSG (*) BYTE DATA  
    ('SYSTEM ALREADY MAINTAINS 15 CONTACTS. $$');  
  
  CALL CRT$PRINT$STRING(.MSG);  
  CALL SEND$CRLF;  
  CALL SEND$CRLF;  
  CALL CHECK$GO$KEY;  
  CALL CLEAR$LOW$SCREEN;  
  END TOO$MANY$CONTACTS;  
*/
```

MAIN\$MODULE

MOVE\$OWN\$SHIP

```

/*****
*
* MOVE$OWN$SHIP:
* THIS PROCEDURE IS EXECUTED EACH TIME A PREDETERMINED PERIOD OF TIME ELAP-
* SES. IT IS USED TO CALCULATE THE MOVEMENTS OF THE SHIP DURING THAT PERIOD.
*
*****/
MOVE$OWN$SHIP: PROCEDURE PUBLIC;
    DCL DELTA$X (4) BYTE,
    DELTA$Y (4) BYTE,
    COS (4) BYTE,
    SIN (4) BYTE,
    TIME (4) BYTE,
    TIME$FLOAT (4) BYTE,
    SPEED (4) BYTE,
    COURSE (4) BYTE,
    DISTANCE (4) BYTE,
    S ADDRESS,
    (1, POINTER, LAST$INFO, H, M) BYTE,

    DCL FP$3600 (4) BYTE DATA (00H, 00H, 61H, 45H), /* 3600.00 */
    DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH), /* 0.0174532925 */

    /* SAVE TIME OF CALL */
    H, M, S = 00H;
    S = TIME$STEP;
    TIME$STEP = 00H;
    TIME(1), TIME(2), TIME(3) = 00H;
    TIME(0) = LOW(S);
    IF S > 255
    THEN TIME(1) = HIGH(S);

```

MAIN\$MODULE

MOVE\$OWN\$SHIP

```

/* FIND INTERVAL OF TIME PAST */
DO WHILE S >= 60;
  S = S - 60;
  M = M + 1;
END;
/* CONVERT TIME TO FP FORMAT */
CALL FLTDS(.TIME, .TIME$FLOAT);
/* CONVERT SPEED IN KNOTS INTO MILES/SECONDS */
POINTER = OWN$SHIP$INFO.POINTER;
CALL FDIV(.OWN$SHIP(POINTER).SPD, .FP$3600, .SPEED);
/* CONVERT COURSE TO ANGLE IN RADIAN */
CALL FMUL(.OWN$SHIP(POINTER).CRS, .DEG$TO$RAD, .COURSE);
/* GET SINE AND COSINE VALUES */
CALL COS$SIN(.COURSE, .COS, .SIN);
/* FIND VARIATIONS IN X AND Y PARAMETERS */
CALL FMUL(.TIME$FLOAT, .SPEED, .DISTANCE);
CALL FMUL(.DISTANCE, .SIN, .DELTA$X);
CALL FMUL(.DISTANCE, .COS, .DELTA$Y);
/* UPDATE OWN SHIP VALUES */
OWN$SHIP$INFO.POINTER = OWN$SHIP$INFO.POINTER + 1;
IF OWN$SHIP$INFO.POINTER = 30
  THEN DO;
    OWN$SHIP$INFO.POINTER = 0;
    OWN$SHIP$INFO.FLAG = TRUE;
  END;
LAST$INFO = POINTER;
POINTER = OWN$SHIP$INFO.POINTER;
OWN$SHIP(POINTER).TIME(0) = OWN$SHIP(LAST$INFO).TIME(0) + H;
OWN$SHIP(POINTER).TIME(1) = OWN$SHIP(LAST$INFO).TIME(1) + M;
OWN$SHIP(POINTER).TIME(2) = OWN$SHIP(LAST$INFO).TIME(2) + S;
DO WHILE OWN$SHIP(POINTER).TIME(2) >= 60;

```


MAIN\$MODULE

MOVE\$OWN\$SHIP

```

OWN$SHIP<POINTER>.TIME(2) = OWN$SHIP<POINTER>.TIME(2) - 60;
OWN$SHIP<POINTER>.TIME(1) = OWN$SHIP<POINTER>.TIME(1) + 1;
END;

DO WHILE OWN$SHIP<POINTER>.TIME(1) >= 60;
  OWN$SHIP<POINTER>.TIME(1) = OWN$SHIP<POINTER>.TIME(1) - 60;
  OWN$SHIP<POINTER>.TIME(0) = OWN$SHIP<POINTER>.TIME(0) + 1;
END;

IF OWN$SHIP<POINTER>.TIME(0) >= 24
  THEN OWN$SHIP<POINTER>.TIME(0) = OWN$SHIP<POINTER>.TIME(0) - 24;
  CALL FADD< OWN$SHIP<LAST$INFO>$.X, .DELTA$, .OWN$SHIP<POINTER>$.X>;
  CALL FADD< OWN$SHIP<LAST$INFO>$.Y, .DELTA$, .OWN$SHIP<POINTER>$.Y>;
  CALL CONV$LAT$LONG< OWN$SHIP<POINTER>$.X, .OWN$SHIP<POINTER>$.Y,
    .OWN$SHIP$INFO.LAT, .OWN$SHIP$INFO.LONG>;
DO I = 0 TO 3;
  OWN$SHIP<POINTER>.CRS(I) = OWN$SHIP<LAST$INFO>.CRS(I);
  OWN$SHIP<POINTER>.SPD(I) = OWN$SHIP<LAST$INFO>.SPD(I);
END;
/* DISPLAY ACTUAL VALUES. */
CALL LAT$LONG$FORMAT< OWN$SHIP$INFO.LAT, .LAT$STRING, 0>;
CALL LAT$LONG$FORMAT< OWN$SHIP$INFO.LONG, .LONG$STRING, 1>;
CALL PRINT$LAT$LONG< LAT$STRING, .LONG$STRING>;
/* DRAW NEW POSITION IN PLASMA DISPLAY. */
CALL PLASMA$OS;
/* ALL DONE. RETURN */
END MOVE$OWN$SHIP;

```

MAIN\$MODULE

EXECUTIVE

```

/**** EXECUTIVE: ****/

```

```

EXECUTIVE: DO;

```

```

TIME$STEP = 00H;
TIME$LIMIT = 180;
WIND$FLAG = FALSE;
CALL CLEAR$STRUCTURES;
CALL INITIALIZE$PLASMA;
CALL GET$SYSTEM$PARAMETERS;
CALL SET$WINDOW;
CALL CLEAR$PLASMA;
CALL DISPLAY$PLASMA$SCALE;
CALL PUT$OS$CENTER;
CALL PLASMA$OS;

```

```

DO FOREVER;
  IF SEC$TIME      /* A SECOND HAS ELAPSED. DISPLAY ACTUAL TIME */
  THEN DO;
    SEC$TIME = FALSE;
    CALL ACTUAL$TIME;
    CALL PRINT$TIME(.TIME$BUFFER);
    CALL CRT$WRITE(PROMPT);
    END;

```

```

IF TIME$STEP >= TIME$LIMIT
  THEN CALL MOVE$OWN$SHIP; /* TIME TO UPDATE OWN SHIP POSITION */

```

MAIN\$MODULE

EXECUTIVE

```

COMMAND = CRT$TRY$READ;      /* CHECK FOR INPUT FROM KEYBOARD */

IF (COMMAND < DISPLAY$UPPER$LIMIT) AND
  (COMMAND > 0)              /* CHECK FOR DISPLAY TYPE COMMANDS */
THEN DO;
  CALL PRINT$MODE(. DISPLAY);
  DO CASE COMMAND;
    ;                      /* CASE 00H */
    DO;                    /* CASE 01H */
      IF WIND$FLAG
      THEN CALL DISPLAY$WIND;
      ELSE CALL NO$WIND;
    END;
    ;                      /* CASE 02H */
    ;                      /* CASE 03H */
    DO;                    /* CASE 04H */
      CALL DISPLAY$SAFE$RNG;
    END;
    ;                      /* CASE 05H */
    ;                      /* CASE 06H */
    DO;                    /* CASE 06H */
      CALL DISPLAY$UPDATE$TIME(TIME$LIMIT);
    END;
  
```

MAIN\$MODULE

EXECUTIVE

```

; /* CASE 07H */
; /* CASE 08H */
; /* CASE 09H */
; /* CASE 0AH */
DO; /* CASE 0BH */
  IF SYSTEM.NUMCTS > 0
    THEN CALL DISPLAY$CONTACT$INFO;
  ELSE CALL NO$CONTACT;
END;
DO; /* CASE 0CH */
  CALL DISPLAY$ORIGIN;
END;
; /* CASE 0DH */
DO; /* CASE 0EH */
  CALL DISPLAY$SCALE;
END;
DO; /* CASE 0FH */
  CALL DISPLAY$OWN$SHIP;
END;
; /* CASE 10H */
; /* CASE 11H */

```


MAIN\$MODULE

EXECUTIVE

```

; /* CASE 12H */
; /* CASE 13H */
; /* CASE 14H */
; /* CASE 15H */
; /* CASE 16H */
; /* CASE 17H */
; /* CASE 18H */

DO; /* CASE 19H */
  IF SYSTEM.NUMCTS > 0
    THEN CALL DISPLAY$SYSTEM;
    ELSE CALL NO$CONTACT;
  END;

END; /* END CASE */

CALL PRINT$MODE(. INPUT);
END; /* END THEN DO */

IF (COMMAND > INPUT$LOWER$LIMIT) AND (COMMAND < INPUT$UPPER$LIMIT)
THEN DO; /* INPUT COMMAND DETECTED */
DO CASE (COMMAND - 2EH);
DO; /* CASE 2EH */
  CALL REORIENT$PS;
END;

```

MAIN\$MODULE

EXECUTIVE

```

DO; /* CASE 2FH */
  IF SYSTEM.NUMCTS > 0
    THEN CALL UPDATE;
  ELSE CALL NO$CONTACT;
END;

DO; /* CASE 30H */
  CALL GET$SAFE$RNG;
END;

DO; /* CASE 31H */
  IF SYSTEM.NUMCTS > 0
    THEN CALL REDESIGNATE;
  ELSE CALL NO$CONTACT;
END;

DO; /* CASE 32H */
  IF (TEMP := INPUT$TIME) <> 0
    THEN TIME$LIMIT = TEMP;
END;

DO; /* CASE 33H */
  CALL SCALE;
END;

DO; /* CASE 34H */
  IF SYSTEM.NUMCTS > 0
    THEN CALL REMOVE;
  ELSE CALL NO$CONTACT;
END;

```

MAIN\$MODULE

EXECUTIVE

```

DO;
  /* CASE 35H */
  IF (SYSTEM.NUMCTS > 6)
    THEN CALL SWAP$CONTACTS;
  ELSE DO;
    IF SYSTEM.NUMCTS < 0
      THEN CALL NOT$ENOUGH$CONTACTS;
    ELSE CALL NO$CONTACT;
  END;
END;

DO;
  /* CASE 36H */
  CALL WIND;
  WIND$FLAG = TRUE;
END;

DO;
  /* CASE 37H */
  IF SYSTEM.NUMCTS < 15
    THEN DO;
      CALL CREATE;
      SYSTEM.NUMCTS = SYSTEM.NUMCTS + 1;
    END;
  ELSE CALL TOO$MANY$CONTACTS;
END;

DO;
  /* CASE 38H */
  CALL OWN$SHIP$UPDATE;
END;

DO;
  /* CASE 39H */
  CALL ORIGIN;

```

AD-A059 603

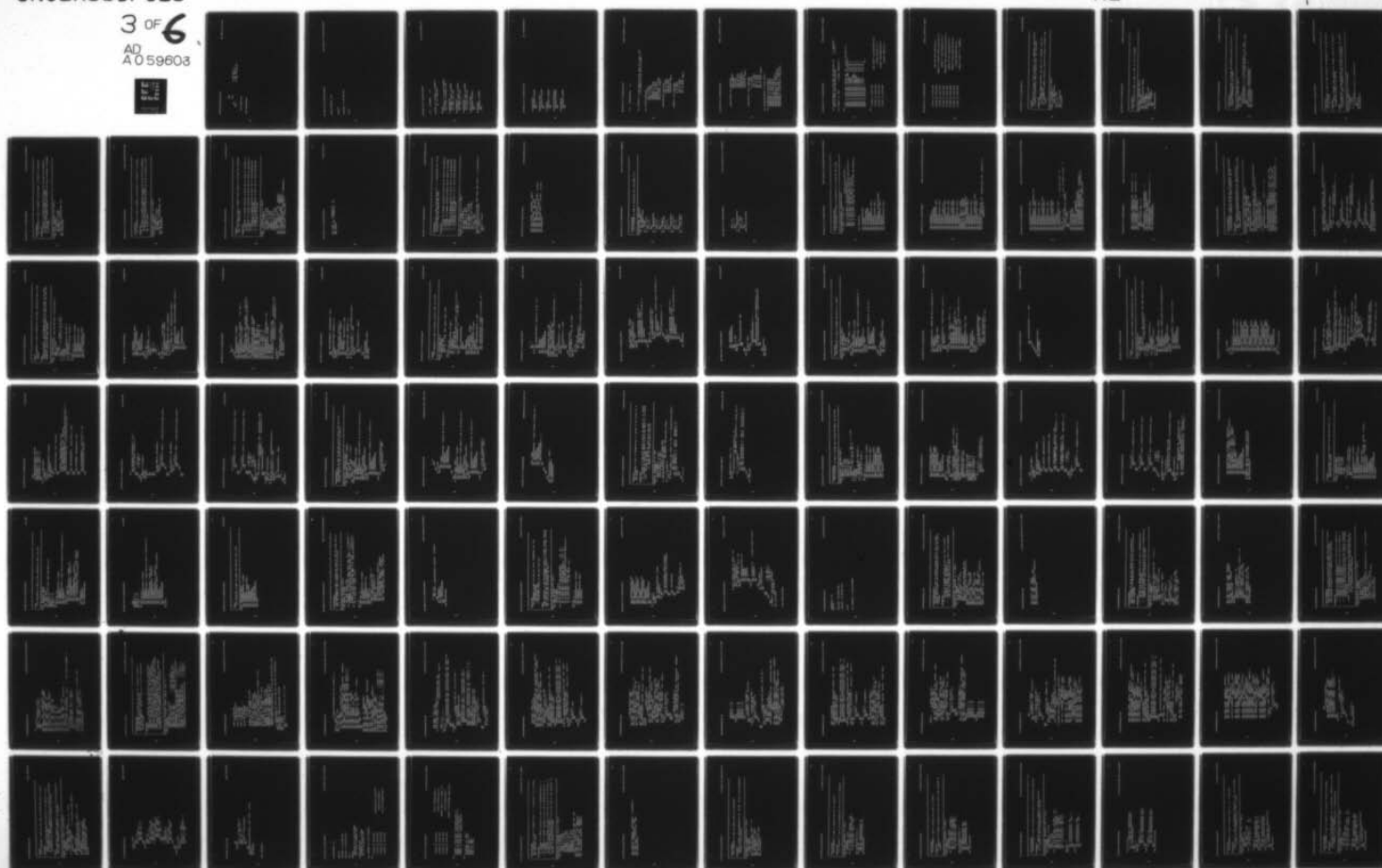
NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF
A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOT--ETC(U)
JUN 78 A L GONCALVES, J E CUBA BRAVO

F/G 17/7

UNCLASSIFIED

NL

3 of 6
AD
A059603



1 F 1 E D

3 OF 6

AD
A 0 5 9 6 0 3



MAIN\$MODULE

EXECUTIVE

END;

END;

END;

END;

/* END CASE */
/* END THEN DO */
/* END DO FOREVER */

END EXECUTIVE

END MAIN\$MODULE;

EXECUTIVE#CMDS

EXECUTIVE#CMDS

EXECUTIVE#CMDS: DO;

\$NOLIST

\$INCLUDE (:F1:EXTER.SRC)

\$LIST

EXECUTIVE\$CMDS

EXTERNALS

```
/*** EXTERNALS:   ***/
```

```
DECLARE LIT LITERALLY 'LITERALLY',  
        DCL LIT 'DECLARE',
```

```
GET$CPA:  
  PROCEDURE (A, B) BYTE EXTERNAL;  
  DCL A BYTE, B ADDRESS; END;
```

```
PLASMA#REDESIG:  
  PROCEDURE (INDEX) EXTERNAL;  
  DCL INDEX BYTE; END;
```

```
PLASMA#DELETE:  
  PROCEDURE (INDEX) EXTERNAL;  
  DCL INDEX BYTE; END;
```

```
PLASMA#CONTACT:  
  PROCEDURE (INDEX) EXTERNAL;  
  DCL INDEX BYTE; END;
```

```
CLEAR$STRUCTURES:  
  PROCEDURE EXTERNAL;  
  END;
```

```
SET$WINDOW:  
  PROCEDURE EXTERNAL;  
  END;
```


EXECUTIVE#CMDS

```
PUT$OS$CENTER:  
  PROCEDURE EXTERNAL;  
END;
```

```
DRAW$EVERYTHING:  
  PROCEDURE EXTERNAL;  
END;
```

```
PLASMA$OS:  
  PROCEDURE EXTERNAL;  
END;
```

```
DISPLAY$PLASMA$SCALE:  
  PROCEDURE EXTERNAL;  
END;
```

```
MOVE$OWN$SHIP:  
  PROCEDURE EXTERNAL;  
END;
```

EXTERNALS

EXECUTIVE#CMDS

DECLARATIONS

/*** DECLARATIONS: ***/

DCL SAFE#RNG (4) BYTE PUBLIC INITIAL (08H, 3CH, 0CAH, 03CH);
/* 0.0246868278 MILES OR FIFTY YARDS */

DCL SYSTEM STRUCTURE

(LAT (4) BYTE,
LONG (4) BYTE,
SCALE (4) BYTE,
WIND\$DIR (4) BYTE,
WIND\$SPD (4) BYTE,
NUM\$ZONE (5) BYTE,
CONTACT\$KIND (3) BYTE,
NUMCTS BYTE) PUBLIC,

OWN\$SHIP\$INFO STRUCTURE

(LAT (4) BYTE,
LONG (4) BYTE,
POINTER BYTE,
FLAG BYTE) PUBLIC,

OWN\$SHIP (30) STRUCTURE

(X (4) BYTE,
Y (4) BYTE,
TIME (3) BYTE,
CRS (4) BYTE,
SPD (4) BYTE) PUBLIC,

EXECUTIVE#CMDS

DECLARATIONS

```
CONTACT$INFO (15) STRUCTURE
  (DESIG ADDRESS,
  TYPE BYTE,
  KIND BYTE,
  CRS$FLAG BYTE,
  SPD$FLAG BYTE,
  OS$POINTER BYTE,
  POINTER BYTE,
  FLAG BYTE) PUBLIC,
```

```
CONTACT$POS1 (225) STRUCTURE
  (X (4) BYTE,
  Y (4) BYTE,
  TIME (3) BYTE,
  CRS (4) BYTE,
  SPD (4) BYTE,
  BRG (4) BYTE,
  RNG (4) BYTE) PUBLIC;
```

```
DCL CONTACT$DISPLAY (6) BYTE PUBLIC;

DCL LAT$STRING (9) BYTE PUBLIC,
LONG$STRING (9) BYTE PUBLIC,
CRS$STRING (6) BYTE PUBLIC,
SPD$STRING (5) BYTE PUBLIC,
CONTACTS$STRING (8) BYTE PUBLIC,
CONTACT$INFO$STRING (44) BYTE PUBLIC,
INPUT$MODE (*) BYTE DATA (' INPUT $$'),
DISPLAY$MODE (*) BYTE DATA ('DISPLAY$$');
```

EXECUTIVE\$CMDS

DECLARATIONS

```
DCL FP$MIN$TO$RAD (4) BYTE DATA (98H,82H,98H,39H), /* 0.00029089 */
FP$2 (4) BYTE DATA (00H,00H,00H,40H); /* 2.0 */
```

```
DCL PROMPT LIT '25H'; /* PROMPT CHARACTER */
```

```
DCL MSG$0 (*) BYTE DATA ('PRESS THE 'GO'' KEY TO CONTINUE:$$',
MSG$1 (*) BYTE DATA ('DO YOU NEED TO UPDATE:$$',
MSG$2 (*) BYTE DATA ('COURSE? (Y/N) $$',
MSG$3 (*) BYTE DATA ('SPEED? (Y/N) $$',
MSG$4 (*) BYTE DATA ('LATITUDE? (Y/N) $$',
MSG$5 (*) BYTE DATA ('LONGITUDE? (Y/N) $$',
MSG$6 (*) BYTE DATA ('BEARING? (Y/N) $$',
MSG$7 (*) BYTE DATA ('RANGE? (Y/N) $$',
MSG$8 (*) BYTE DATA ('DESIG? (Y/N) $$',
MSG$9 (*) BYTE DATA ('TYPE? (Y/N) $$',
MSG$A (*) BYTE DATA ('CLASS? (Y/N) $$',
BLANK (*) BYTE DATA (' $$/);
```

```
DCL TITLE$0 (*) BYTE DATA
(' NEW CONTACT INITIALIZATION. $$/),
TITLE$1 (*) BYTE DATA
(' CONTACT REMOVAL. $$/),
TITLE$2 (*) BYTE DATA
(' CONTACT REDESIGNATION. $$/),
TITLE$3 (*) BYTE DATA
(' CONTACT UPDATE. $$/),
```


EXECUTIVE#CMDS

TITLE\$4 (*) BYTE DATA
(
TITLE\$5 (*) BYTE DATA
(
TITLE\$6 (*) BYTE DATA
(
TITLE\$9 (*) BYTE DATA
(
TITLE\$A (*) BYTE DATA
(
TITLE\$B (*) BYTE DATA
(
TITLE\$C (*) BYTE DATA
(
TITLE\$D (*) BYTE DATA
(

DECLARATIONS

OWN SHIP DATA UPDATING. \$\$'),
CHANGE OF CONTACTS BEING DISPLAYED. \$\$'),
CHANGE OF TIME PARAMETERS. \$\$'),
COORDINATE GRID ORIGIN MODIFICATION. \$\$'),
CHANGE OF WIND INFORMATION. \$\$'),
GRAPHICS SCALE MODIFICATION. \$\$'),
PICTURE REORIENTATION. \$\$'),
CHANGE OF SAFE C. P. A. RANGE\$\$')

EXECUTIVE\$CMDS

DE\$HASH

```
/******  
* DE$HASH:  
* THIS PROCEDURE IS USED TO GET A PAIR OF ASCII CHARACTERS, REPRESENTING  
* A CONTACT'S DESIGNATION, FROM AN ADDRESS VALUE. SEE 'GET$DESIG'.  
*  
* PARAMETERS:  
* - A - ADDRESS VALUE CONTAINING THE CODE TO BE 'DEHASHED' INTO TWO  
* ASCII CHARACTERS.  
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO BE  
* PLACED.  
*  
*****/  
DE$HASH: PROCEDURE (A,B) PUBLIC;  
  DCL (A,B) ADDRESS;  
  CHAR BASED B BYTE;  
  CHAR = A / 100;  
  B = B + 1;  
  CHAR = A MOD 100;  
  END DE$HASH;
```

EXECUTIVE#CMDS

CHECK\$GO\$KEY

```
/*  
*  
* CHECK$GO$KEY:  
* THIS PROCEDURE IS USED TO CHECK IF THE 'GO' KEY IS PRESSED.  
*  
*****  
CHECK$GO$KEY: PROCEDURE PUBLIC;  
  DCL CHAR BYTE;  
  CALL CRT$PRINT$STRING(,MSG$0);  
  CALL SEND$BEL;  
  CHAR = CRT$READ;  
  DO WHILE CHAR <> 02CH;  
    CALL SEND$BEL;  
    CHAR = CRT$READ;  
  END;  
  END CHECK$GO$KEY;  
*****  
*/
```

EXECUTIVE\$CMDS

DISPLAY\$KIND

```
/******  
* DISPLAY$KIND:  
* THIS PROCEDURE IS USED TO DISPLAY THE INFORMATION ABOUT THE KIND OF  
* CONTACTS MAINTAINED BY THE SYSTEM.  
*  
*****/  
DISPLAY$KIND: PROCEDURE PUBLIC;  
  DCL I BYTE;  
  DO I = 0 TO 2;  
    CONTACTS$STRING(2 * I) = SYSTEM.CONTACT$KIND(1) / 10 + 30H;  
    CONTACTS$STRING((2 * I) + 1) = SYSTEM.CONTACT$KIND(1) MOD 10 + 30H;  
  END;  
  CONTACTS$STRING(6), CONTACTS$STRING(7) = '$';  
  CALL PRINT$CONTACT$(CONTACTS$STRING);  
  END DISPLAY$KIND;
```


EXECUTIVE\$CMDS

CHECK\$DESIG

```
/******  
* CHECK$DESIG:  
* THIS PROCEDURE IS USED TO USED TO DETECT THE PRESENCE OF A GIVEN CONTACT  
* IN THE SYSTEM.  
*  
* PARAMETERS:  
* - A - ADDRESS VARIABLE THAT CONTAINS THE 'HASHED' VALUE OF THE CONTACT'S  
* DESIGNATION DESIRED TO BE CHECKED.  
*  
* USAGE:  
* TYPED PROCEDURE. A VALUE INDICATING THE RELATIVE POSITION OF THE CONTACT  
* IS RETURNED IF FOUND. OTHERWISE A VALUE OF 0FFH IS RETURNED.  
*  
*****/  
CHECK$DESIG: PROCEDURE (A) BYTE PUBLIC;  
  DCL A ADDRESS,  
  I BYTE;  
  DO I = 0 TO 14;  
    IF CONTACT$INFO(I).DESIG = A THEN RETURN I;  
  END;  
  RETURN 0FFH;  
END CHECK$DESIG;
```

EXECUTIVE\$CMDS

CONV\$MIN\$RAD

```
/******  
* CONV$MIN$RAD:  
* THIS PROCEDURE IS USED TO CONVERT A GIVEN ANGLE, IN MINUTES, TO RADIANS.  
*  
* PARAMETERS:  
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESENTATION OF AN ANGLE IN MINUTES, IS LOCATED.  
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE IN RADIANS IS DESIRED TO BE PLACED.  
*  
*****/  
CONV$MIN$RAD: PROCEDURE (A,B) PUBLIC;  
  DCL (A,B) ADDRESS,  
       MIN BASED A (4) BYTE,  
       RAD BASED B (4) BYTE;  
  CALL FMUL< MIN, FP$MIN$TO$RAD, .RAD>;  
  END CONV$MIN$RAD;
```

EXECUTIVE\$CMDS

CONV\$RAD\$MIN

```
/******  
* CONV$RAD$MIN:  
* THIS PROCEDURE IS USED TO CONVERT A GIVEN ANGLE, IN RADIANS, TO MINUTES.  
*  
* PARAMETERS:  
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESENTATION OF AN ANGLE IN RADIANS, IS LOCATED.  
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE IN MINUTES IS DESIRED TO BE PLACED.  
*  
*****/  
CONV$RAD$MIN: PROCEDURE (A,B) PUBLIC;  
  DCL (A,B) ADDRESS,  
  RAD BASED A (4) BYTE,  
  MIN BASED B (4) BYTE;  
  CALL FDIVC, RAD, FP$MIN$TO$RAD, MIN;  
  END CONV$RAD$MIN;
```

EXECUTIVE#CMDS

CONV\$XY

```

/*****
*
* CONV$XY:
* THIS PROCEDURE IS USED TO CONVERT GIVEN VALUES OF LATITUDE AND LONGITUDE
* INTO 'X,Y' COORDINATES.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF LATITUDE IS
*   LOCATED.
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF LONGITUDE IS
*   LOCATED.
* - C - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*   'X' IS DESIRED TO BE PLACED.
* - D - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*   'Y' IS DESIRED TO BE PLACED.
*
*****/
CONV$XY: PROCEDURE (A,B,C,D) PUBLIC;
  DCL (A,B,C,D) ADDRESS,
    LAT BASED A (4) BYTE,
    LONG BASED B (4) BYTE,
    X BASED C (4) BYTE,
    Y BASED D (4) BYTE,
    MEAN$LAT (4) BYTE,
    COS$MEAN$LAT (4) BYTE,
    SIN$MEAN$LAT (4) BYTE;
  CALL FADD< .SYSTEM.LAT, .LAT, .MEAN$LAT>;
  CALL FDIV< .MEAN$LAT, .FP$2, .MEAN$LAT>;
  CALL CONV$MIN$RAD< .MEAN$LAT, .MEAN$LAT>;
  CALL COS$SIN< .MEAN$LAT, .COS$MEAN$LAT, .SIN$MEAN$LAT>;
  CALL FSUB< .LONG, .SYSTEM.LONG, .X>;

```


CONV\$XY

EXECUTIVE\$CMDS

CALL FMUL(C,X),COS\$MEAN\$LAT, .X)
CALL FSUB(C,LAT, .SYSTEM.LAT, .Y)
END CONV\$XY

EXECUTIVE\$CMDS

CONV\$REL\$XY

```

/*****
* CONV$REL$XY:
* THIS PROCEDURE IS USED TO OBTAIN VALUES OF 'X,Y' COORDINATES FOR A POINT
* TO WHICH A RANGE AND BEARING ARE GIVEN.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF THE BEARING
*   IS LOCATED.
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE OF THE RANGE
*   IS LOCATED.
* - C - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*   'X' IS DESIRED TO BE PLACED.
* - D - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*   'Y' IS DESIRED TO BE PLACED.
*
*****
CONV$REL$XY: PROCEDURE (A,B,C,D) PUBLIC;
    DCL (A,B,C,D) ADDRESS,
        BRG BASED A (4) BYTE,
        RNG BASED B (4) BYTE,
        DELTA$X BASED C (4) BYTE,
        DELTA$Y BASED D (4) BYTE,
        COS (4) BYTE, SIN (4) BYTE,
        ANGLE (4) BYTE,
        (I, TEMP) BYTE;
    DCL DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH); /* 0.0174532925 */
    TEMP = OWN$SHIP$INFO.POINTER;
    DO I = 0 TO 3;
        ANGLE(I) = BRG(I);
    END;

```

EXECUTIVE\$CMDS

CONV\$REL\$XY

```
CALL FMUL<.ANGLE, .DEG$TO$RAD, .ANGLE>;
CALL COS$SIN<.ANGLE, .COS, .SIN>;
CALL FMUL<.RNG, .SIN, .DELTA$X>;
CALL FADD<.OWN$SHIP<TEMP>, X, .DELTA$X, .DELTA$X>;
CALL FMUL<.RNG, .COS, .DELTA$Y>;
CALL FADD<.OWN$SHIP<TEMP>, Y, .DELTA$Y, .DELTA$Y>;
END CONV$REL$XY;
```

EXECUTIVE\$CMDS

INIT\$STRUCTURES

```
/******  
*  
* INIT$STRUCTURES:  
* THIS PROCEDURE IS USED TO INITIALIZE ALL STRUCTURES TO 0.  
*  
*****  
INIT$STRUCTURES: PROCEDURE;  
DCL (A,I) ADDRESS;  
A = .SYSTEM;  
DO I = 0 TO 28;  
TEMP = 0;  
A = A + 1;  
END;  
A = .OWN$SHIP$INFO;  
DO I = 0 TO 9;  
TEMP = 0;  
A = A + 1;  
END;  
A = .OWN$SHIP;  
DO I = 0 TO 569;  
TEMP = 0;  
A = A + 1;  
END;  
A = .CONTACT$INFO;  
DO I = 0 TO 134;  
TEMP = 0;  
A = A + 1;  
END;  
A = .CONTACT$POS1;  
DO I = 0 TO 6074;
```


INIT\$STRUCTURES

EXECUTIVE\$CMDS

```
TEMP = 0;  
A = A + 1;  
END;  
A = . CONTACT$DISPLAY;  
DO I = 0 TO 5;  
TEMP = OFFH;  
A = A + 1;  
END;  
END INIT$STRUCTURES;
```

EXECUTIVE\$CMDS

GET\$SYSTEM\$PARAMETERS

```

/*****
*
* GET$SYSTEM$PARAMETERS:
*   THIS PROCEDURE IS USED BY THE EXECUTIVE TO INITIALIZE THE SYSTEM.
*
*****/
GET$SYSTEM$PARAMETERS: PROCEDURE PUBLIC;
  DCL MSG0 (*) BYTE DATA
    (
      MSG1 (*) BYTE DATA (<'TIME INITIALIZATION:$$',
      MSG2 (*) BYTE DATA (<'COORDINATE GRID ORIGIN INITIALIZATION:$$',
      MSG3 (*) BYTE DATA (<'OWN SHIP INITIAL DATA:$$',
      MSG4 (*) BYTE DATA (<'INITIAL GRAPHICS SCALE:$$',
      MSG5 (*) BYTE DATA (<'PRESS THE ''GO'' KEY TO START THE SYSTEM.$$',
    )
  SYSTEM INITIALIZATION:$',
  )
  DCL CHAR BYTE;

  CALL CRT$MASTER$CLEAR;
  CALL INIT$FP;
  CALL INIT$STRUCTURES;
  CALL INIT$HIGH$SCREEN;
  CALL SET$LOW$HOME;
  CALL CRT$PRINT$STRING(< MSG0);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(< MSG1);
  CALL SEND$CRLF;
  CALL GET$TIME$ZONE(< SYSTEM.NUM$ZONE);
  CALL CRT$PRINT$STRING(< MSG0);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(< MSG1);

```

EXECUTIVE\$CMDS

GET\$SYSTEM\$PARAMETERS

```
CALL SEND$CRLF;
CALL INITIATE$TIME;
CALL CLEAR$LOW$SCREEN;
CALL CRT$PRINT$STRING(. MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG2);
CALL SEND$CRLF;
CALL GET$LAT(. SYSTEM. LAT);
CALL CRT$PRINT$STRING(. MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG2);
CALL SEND$CRLF;
CALL GET$LONG(. SYSTEM. LONG);
CALL CRT$PRINT$STRING(. MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG3);
CALL SEND$CRLF;
OWN$SHIP(0). TIME(0) = HOURS;
OWN$SHIP(0). TIME(1) = MINUTES;
OWN$SHIP(0). TIME(2) = SECONDS;
CALL GET$LAT(. OWN$SHIP$INFO. LAT);
CALL CRT$PRINT$STRING(. MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG3);
CALL SEND$CRLF;
CALL GET$LONG(. OWN$SHIP$INFO. LONG);
CALL CRT$PRINT$STRING(. MSG0);
CALL SEND$CRLF;
OWN$SHIP$INFO. POINTER = 0;
CALL CONV$XY(. OWN$SHIP$INFO. LAT, . OWN$SHIP$INFO. LONG, . OWN$SHIP(0). X,
. OWN$SHIP(0). Y);
```

EXECUTIVE\$CMDS

GET\$SYSTEM\$PARAMETERS

```

CALL CRT$PRINT$STRING(.MSG3);
CALL SEND$CRLF;
CALL GET$COURSE$BRG(0, .OWN$SHIP(0).CRS);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG3);
CALL SEND$CRLF;
CALL GET$SPEED(.OWN$SHIP(0).SPD);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG4);
CALL SEND$CRLF;
CALL GET$SCALE(.SYSTEM.SCALE);
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$CRLF;
CALL START$BLINK;
CALL CRT$PRINT$STRING(.MSG5);
CALL CRT$WRITE(18H); /* STOP BLINK MODE */
CHAR = 0;
DO WHILE CHAR <> 2CH; /* WAIT FOR ORDER TO START */
  IF CHAR <> 2CH THEN CALL SEND$BEL;
  CHAR = CRT$READ;
END;
CALL INITIATE$CLOCK;
CALL CLEAR$LOW$SCREEN;
CALL ACTUAL$TIME;
CALL PRINT$TIME(.TIME$BUFFER);
CALL PRINT$TIME$ZONE(.SYSTEM.NUM$ZONE);
CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LAT, .LAT$STRING, 0);
CALL LAT$LONG$FORMAT(.OWN$SHIP$INFO.LONG, .LONG$STRING, 1);
CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);

```


EXECUTIVE#CMDS

GET\$SYSTEM\$PARAMETERS

```
CHAR = FP$FORMAT(.OWN$SHIP(0),CRS, .CRS$STRING, 3, 1);  
CALL PRINT$COURSE(.CRS$STRING);  
CHAR = FP$FORMAT(.OWN$SHIP(0),SPD, .SPD$STRING, 2, 1);  
CALL PRINT$SPEED(.SPD$STRING);  
DO CHAR = 0 TO 5;  
    CONTACTS$STRING(CHAR) = '0';  
END;  
CONTACTS$STRING(6), CONTACTS$STRING(7) = '$';  
CALL PRINT$CONTACTS(.CONTACTS$STRING);  
CALL PRINT$MODE(.INPUT$MODE);  
CONTACT$INFO$STRING(42), CONTACT$INFO$STRING(43) = '$';  
CALL CRT$WRITE(PROMPT);  
END GET$SYSTEM$PARAMETERS;
```

EXECUTIVE#CMDS

DISPLAY#CONTACT

```

/*****
*
* DISPLAY#CONTACT:
* THIS PROCEDURE IS USED TO DISPLAY ALL THE AVAILABLE INFORMATION ABOUT
* A GIVEN CONTACT, ACCORDING TO THE FORMAT PRESENT AT THE CRT.
*
* PARAMETERS:
* - ROW - INDICATES IN WHICH DISPLAY ROW TO PUT THE INFORMATION.
* - INDEX - SHOWS THE RELATIVE POSITION OF THE CONTACT IN THE DATA STRUCTURES BEING USED.
*
*****
*****
DISPLAY#CONTACT: PROCEDURE (ROW, INDEX) PUBLIC
  DCL KIND (*) BYTE DATA ('FRIHOSUNK'),
    (ROW, INDEX, I, J, TEMP, TEST) BYTE;
  CONTACT#DISPLAY(ROW) = INDEX;
  ROW = ROW + 1;
  CALL DE$HASH (CONTACT#INFO(INDEX).DESIG, CONTACT#INFO$STRING(0));
  CONTACT#INFO$STRING(2), CONTACT#INFO$STRING(3) = 'S';
  IF CONTACT#INFO(INDEX).TYPE = 0
    THEN CONTACT#INFO$STRING(2) = ' / ';
  J = CONTACT#INFO(INDEX).KIND;
  DO I = 0 TO 2;
    CONTACT#INFO$STRING(4 + I) = KIND((3 * J) + I);
  END;
  J = CONTACT#INFO(INDEX).POINTER;
  CONTACT#INFO$STRING(7) = CONTACT#POSI(J).TIME(0) / 10 + 30H;
  CONTACT#INFO$STRING(8) = CONTACT#POSI(J).TIME(0) MOD 10 + 30H;
  CONTACT#INFO$STRING(9) = CONTACT#POSI(J).TIME(1) / 10 + 30H;
  CONTACT#INFO$STRING(10) = CONTACT#POSI(J).TIME(1) MOD 10 + 30H;
  TEMP = FP$FORMAT( CONTACT#POSI(J).BRG, CONTACT#INFO$STRING(11), 3, 1);

```

EXECUTIVE\$CMDS

DISPLAY\$CONTACT

```
CALL RANGE$FORMAT(. CONTACT$POSI(J). RNG. . CONTACT$INFO$STRING(15));
TEMP = GET$CPA(INDEX. . CONTACT$INFO$STRING(21));
IF TEMP = 0
THEN DO;
  IF CONTACT$INFO(INDEX). CRS$FLAG
  THEN DO;
    TEST = FP$FORMAT(. CONTACT$POSI(J). CRS.
    . CONTACT$INFO$STRING(21). 3. 1);
    END;
  ELSE DO;
    DO I = 21 TO 24;
      CONTACT$INFO$STRING(I) = ' ';
    END;
  END;
  IF CONTACT$INFO(INDEX). SPD$FLAG
  THEN DO;
    TEST = FP$FORMAT(. CONTACT$POSI(J). SPD.
    . CONTACT$INFO$STRING(25). 2. 1);
    END;
  ELSE DO;
    DO I = 25 TO 27;
      CONTACT$INFO$STRING(I) = ' ';
    END;
  END;
  DO I = 28 TO 41;
    CONTACT$INFO$STRING(I) = ' ';
  END;
  END;
CONTACT$INFO$STRING(42). CONTACT$INFO$STRING(43) = '$';
CALL PRINT$CONTACT$INFO(ROW. . CONTACT$INFO$STRING);
END DISPLAY$CONTACT;
```

EXECUTIVE\$CMDS

CREATE

```

/*****
*
* CREATE:
* THIS PROCEDURE IS USED TO OBTAIN ALL PERTINENT INFORMATION ABOUT A NEW
* CONTACT.
*
* USAGE:
* UNTYPED PROCEDURE. THIS PROCEDURE SHOULD BE CALLED AFTER IT HAS BEEN
* DETERMINED THAT A CONTACT CAN BE ACCEPTED IN THE SYSTEM, AND THAT THE
* NUMBER OF CONTACTS HAS BEEN UPDATED.
*
*****/
CREATE: PROCEDURE PUBLIC;
  DCL STR1 (*) BYTE DATA ('ARE THE FOLLOWING VALUES KNOWN?$$'),
    ARRAY (2) BYTE,
    A ADDRESS,
    (OK, TEMP, I, J, INDEX, H, M, S) BYTE;
  H = HOURS; /* TO SAVE THE TIME OF CALL */
  M = MINUTES;
  S = SECONDS;
  /* UPDATE POSITION OF OWN SHIP */
  CALL MOVE$OWN$SHIP;
  /* GET INITIAL CONTACT VALUES */
  OK = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(. TITLE$0);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(. STR1);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(. MSG$2);
    ARRAY(0) = CHECK$YES$NO;

```


EXECUTIVE\$CMDS

CREATE

```
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.MSG$3);
ARRAY(1) = CHECK$YES$NO;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
TEMP, I = 0;
DO WHILE TEMP = 0;
  IF CONTACT$INFO(1).DESIG = 0
  THEN DO;
    TEMP = 1;
    INDEX = 1;
    END;
    I = I + 1;
  END;
  OK = 0;
  DO WHILE OK = 0; /* GET DESIG */
    CALL CRT$PRINT$STRING(.TITLE$0);
    CALL SEND$CRLF;
    A, CONTACT$INFO(INDEX).DESIG = GET$DESIG;
    DO I = 0 TO 14;
      IF (I <> INDEX) AND (A = CONTACT$INFO(1).DESIG)
      THEN DO;
        CALL CRT$PRINT$STRING(.DESIGNATION ALREADY IN USE.$$');
        CALL SEND$CRLF;
        CALL CHECK$GO$KEY;
        OK = 0;
        CALL CLEAR$LOW$SCREEN;
        GO TO L;
      END;
    END;
  END;
END;
```

EXECUTIVE\$CMDS

CREATE

```

OK = 1;
END;

L: END;
CALL CRT$PRINT$STRING(. TITLE$0);
CALL SEND$CRLF;
CONTACT$INFO(INDEX).TYPE = GET$TYPE; /* GET TYPE */
CALL CRT$PRINT$STRING(. TITLE$0);
CALL SEND$CRLF;
TEMP, CONTACT$INFO(INDEX).KIND = GET$KIND; /* GET KIND */
SYSTEM CONTACT$KIND(TEMP) = SYSTEM CONTACT$KIND(TEMP) + 1;
CALL CRT$PRINT$STRING(. TITLE$0);
CALL SEND$CRLF;
J, CONTACT$INFO(INDEX).POINTER = 15*INDEX;
CONTACT$INFO(INDEX).OS$POINTER = 0FFH;
CONTACT$INFO(INDEX).FLAG = 0;
CONTACT$INFO(INDEX).CRS$FLAG = 0;
CONTACT$INFO(INDEX).SPD$FLAG = 0;
CONTACT$POSI(J).TIME(0) = H;
CONTACT$POSI(J).TIME(1) = M;
CONTACT$POSI(J).TIME(2) = S;
CALL GET$COURSE$BRG(1, CONTACT$POSI(J).BRG);
CALL CRT$PRINT$STRING(. TITLE$0);
CALL SEND$CRLF;
CALL GET$RANGE(. CONTACT$POSI(J).RNG);
CALL CONV$REL$XY(. CONTACT$POSI(J).BRG, CONTACT$POSI(J).RNG,
CONTACT$POSI(J).X, CONTACT$POSI(J).Y);
IF ARRAY(0)
THEN DO;
CONTACT$INFO(INDEX).CRS$FLAG = 1;
CALL CRT$PRINT$STRING(. TITLE$0);
CALL SEND$CRLF;

```

EXECUTIVE\$CMDS

CREATE

```
CALL GET$COURSE$BRG(0, CONTACT$POSI(J), CRS);
END;
ELSE CONTACT$INFO(INDEX), CRS$FLAG = 0;
IF ARRAY(1)
THEN DO;
  CONTACT$INFO(INDEX), SPD$FLAG = 1;
  CALL CRT$PRINT$STRING( TITLE$0);
  CALL SEND$CRLF;
  CALL GET$SPEED( CONTACT$POSI(J), SPD);
  END;
ELSE CONTACT$INFO(INDEX), SPD$FLAG = 0;
I, OK = 0;
DO WHILE (OK = 0) AND ( I <= LAST(CONTACT$DISPLAY));
  IF CONTACT$DISPLAY(I) = 0FFH
  THEN DO;
    OK = 1;
    CALL DISPLAY$CONTACT(I, INDEX);
  END;
  I = I + 1;
END;
CALL DISPLAY$KIND;
CALL PLASMA$CONTACT(INDEX);
END CREATE;
```

EXECUTIVE#CMDS

REMOVE

```

/*****
*
* REMOVE:
* THIS PROCEDURE IS USED TO REMOVE A CONTACT FROM THE SYSTEM.
*
*****
REMOVE: PROCEDURE PUBLIC;
DECL DESIG ADDRESS,
      STRING(4) BYTE,
      (ROW, OK, I, CHAR, TEMP, CLASS, CHECK) BYTE;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(.TITLE$1);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(.
    ( 'ARE YOU SURE YOU WANT TO DELETE A CONTACT? (Y/N) $$' ));
  IF ((CHAR:= CHECK$YES$NO) = 0 ) /* NOT SURE */
  THEN DO;
    CALL CLEAR$LOW$SCREEN;
    RETURN;
  END;
  CALL SEND$CRLF;
  DESIG = GET$DESIG;
  CALL DE$HASH(DESIG, .STRING);
  STRING(2), STRING(3) = '$';
  CALL CRT$PRINT$STRING(.TITLE$1);
  CALL SEND$CRLF;
  IF (TEMP:= CHECK$DESIG(DESIG)) <> 0FFH
  THEN DO;
    CALL CRT$PRINT$STRING(. 'CONTACT TO BE DELETED: $$' );
    CALL CRT$PRINT$STRING(. STRING);

```


EXECUTIVE\$CMDS

REMOVE

```
OK = 1;
END;
ELSE DO;
  CALL CRT$PRINT$STRING( 'DESIG NOT IN USE. $$' );
END;
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
END;
CONTACT$INFO(TEMP).DESIG = 00H;
SYSTEM.NUMCTS = SYSTEM.NUMCTS - 1;
CLASS = CONTACT$INFO(TEMP).KIND;
SYSTEM.CONTACT$KIND(CLASS) = SYSTEM.CONTACT$KIND(CLASS) - 1;
ROW = 0FFH;
DO I = 0 TO 5;
  IF CONTACT$DISPLAY(I) = TEMP
  THEN DO;
    CONTACT$DISPLAY(I) = 0FFH;
    ROW = I;
  END;
END;
CALL PLASMA$DELETE(TEMP);
IF (SYSTEM.NUMCTS > 5) AND (ROW <> 0FFH)
THEN DO;
  OK = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(
      'ENTER THE DESIG OF A CONTACT DESIRED TO BE DISPLAYED: $$' );
    CALL SEND$CRLF;
    DESIG = GET$DESIG;
```

EXECUTIVE\$CMDS

REMOVE

```
CALL DE$HASH(DESIG, ., STRING);
STRING(2), STRING(3) = '$';
TEMP = CHECK$DESIG(DESIG);
IF TEMP = 0FFH
THEN DO;
    CALL CRT$PRINT$STRING(, ('DESIG $$$'));
    CALL CRT$PRINT$STRING(, STRING);
    CALL CRT$PRINT$STRING(, (' NOT IN USE. $$$'));
    CALL SEND$CRLF;
END;
ELSE DO;
    CHECK = 0;
    DO I = 0 TO LAST(CONTACT$DISPLAY);
        IF CONTACT$DISPLAY(I) = TEMP
        THEN DO;
            CALL CRT$PRINT$STRING(, ('CONTACT ALREADY DISPLAYED. $$$'));
            CALL SEND$CRLF;
            CHECK = 1;
            I = LAST(CONTACT$DISPLAY) + 2;
        END;
    END;
    IF CHECK = 0
    THEN DO;
        CALL CRT$PRINT$STRING(, ('CONTACT $$$'));
        CALL CRT$PRINT$STRING(, STRING);
        CALL CRT$PRINT$STRING(, (' WILL BE DISPLAYED. $$$'));
        CALL SEND$CRLF;
        OK = 1;
    END;
    END;
    CALL SEND$CRLF;
```

REMOVE

EXECUTIVE\$CMDS

```
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
END;
IF TEMP <> 0FFH
THEN DO;
  OK = 1;
  CALL DISPLAY$CONTACT(ROW, TEMP);
END;
ELSE DO;
  IF ROW <> 0FFH
  THEN DO;
    DO I = 0 TO 41;
      CONTACT$INFO$STRING(I) = ' ';
    END;
    CONTACT$INFO$STRING(42), CONTACT$INFO$STRING(43) = '$';
    CALL PRINT$CONTACT$INFO(ROW + 1, CONTACT$INFO$STRING);
  END;
END;
CALL DISPLAY$KIND;
END REMOVE;
```

EXECUTIVE\$CMDS

REDESIGNATE

```
/******  
* REDESIGNATE:  
* THIS PROCEDURE IS USED TO CHANGE THE DESIG OF A CONTACT.  
*  
*****  
REDESIGNATE: PROCEDURE PUBLIC;  
  DCL (NEW,OLD) ADDRESS,  
    STR1 (4) BYTE,  
    STR2 (4) BYTE,  
    (TEMP, TEMP1, INDEX, I) BYTE;  
  TEMP = 0FFH;  
  DO WHILE TEMP = 0FFH;  
    CALL CRT$PRINT$STRING(, TITLE$2);  
    CALL SEND$CRLF;  
    CALL CRT$PRINT$STRING(, 'ENTER OLD DESIG AS REQUESTED:$$');  
    CALL SEND$CRLF;  
    OLD = GET$DESIG;  
    TEMP = CHECK$DESIG(OLD);  
    IF TEMP = 0FFH  
      THEN DO;  
        CALL CRT$PRINT$STRING(, 'DESIG NOT IN USE. $$');  
        CALL SEND$CRLF;  
        CALL CHECK$GO$KEY;  
        CALL CLEAR$LOW$SCREEN;  
      END;  
    END;  
    TEMP1 = 0;  
    DO WHILE TEMP1 <> 0FFH;  
      CALL CRT$PRINT$STRING(, TITLE$2);  
      CALL SEND$CRLF;
```


EXECUTIVE\$CMDS

REDESIGNATE

```
CALL CRT$PRINT$STRING(, ('ENTER NEW DESIG AS REQUESTED:$$'));  
CALL SEND$CRLF;  
NEW = GET$DESIG;  
TEMP1 = CHECK$DESIG(NEW);  
IF TEMP1 <> 0FFH  
THEN DO;  
CALL CRT$PRINT$STRING(, ('DESIG ALREADY IN USE:$$'));  
CALL SEND$CRLF;  
END;  
ELSE DO;  
STR1(2), STR1(3), STR2(2), STR2(3) = '$';  
CALL DE$HASH(OLD, .STR1);  
CALL DE$HASH(NEW, .STR2);  
CALL CRT$PRINT$STRING(, ('CONTACT $$'));  
CALL CRT$PRINT$STRING(, STR1);  
CALL CRT$PRINT$STRING(, (' WILL BE CHANGED TO $$'));  
CALL CRT$PRINT$STRING(, STR2);  
CALL CRT$WRITE(, ');  
CALL SEND$CRLF;  
CONTACT$INFO(TEMP1).DESIG = NEW;  
CALL SEND$CRLF;  
END;  
CALL CHECK$GO$KEY;  
CALL CLEAR$LOW$SCREEN;  
IF TEMP1 = 0FFH  
THEN DO;  
DO I = 0 TO LAST(CONTACT$DISPLAY);  
IF CONTACT$DISPLAY(I) = TEMP  
THEN DO;  
INDEX = CONTACT$DISPLAY(I);  
CONTACT$INFO(INDEX).DESIG = NEW;
```

EXECUTIVE#CMDS

REDESIGNATE

```
CALL PRINT$CONTACT$INFO(I + 1, .STR2);  
END;  
END;  
END;  
CALL PLASMA$REDESIG(TEMP1);  
END REDESIGNATE;
```

EXECUTIVE\$CMDS

UPDATE

```

/*****
*
* UPDATE:
* THIS PROCEDURE IS USED TO UPDATE INFORMATION ABOUT ANY CONTACT.
*
*****
UPDATE: PROCEDURE PUBLIC;
DECL DESIG ADDRESS,
      ARRAY (6) BYTE,
      (LAST$INFO, OK, I, J, K, TEMP, KIND, OLD$KIND, TYPE, INDEX, COUNT, H, M, S) BYTE;
H = HOURS; /* SAVE TIME OF CALL */
M = MINUTES;
S = SECONDS;
/* UPDATE OWN SHIP POSITION */
CALL MOVE$OWN$SHIP;
/* GET CONTACT VALUES */
OK = 0FFH;
DO WHILE OK = 0FFH;
  CALL CRT$PRINT$STRING(, TITLE$3);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(, 'ENTER CONTACT DESIG AS REQUESTED:$$');
  CALL SEND$CRLF;
  DESIG = GET$DESIG;
  INDEX = CHECK$DESIG(DESIG);
  IF INDEX = 0FFH
  THEN DO;
    CALL CRT$PRINT$STRING(, 'DESIG NOT IN USE. $$');
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
  END;

```

EXECUTIVE#CMDS

UPDATE

```
OK = INDEX;
END;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(. TITLE$3);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG$1);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG$9);
  ARRAY (0) = CHECK$YES$NO;
  CALL CRT$PRINT$STRING(. BLANK);
  CALL CRT$PRINT$STRING(. MSG$A);
  ARRAY (1) = CHECK$YES$NO;
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG$6);
  ARRAY (2) = CHECK$YES$NO;
  CALL CRT$PRINT$STRING(. BLANK);
  CALL CRT$PRINT$STRING(. MSG$7);
  ARRAY (3) = CHECK$YES$NO;
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG$2);
  ARRAY (4) = CHECK$YES$NO;
  CALL CRT$PRINT$STRING(. BLANK);
  CALL CRT$PRINT$STRING(. MSG$3);
  ARRAY (5) = CHECK$YES$NO;
  CALL SEND$CRLF;
  OK = CHECK$INPUT;
  CALL CLEAR$LOW$SCREEN;
END;
TEMP = 0;
DO I = 0 TO LAST(ARRAY);
```


EXECUTIVE\$CMDS

UPDATE

```

IF ARRAY(1) THEN TEMP = 1;
END;
IF TEMP = 0 THEN RETURN; /* NO INPUT IS DESIRED */
J, LAST$INFO = CONTACT$INFO(INDEX). POINTER;
IF ARRAY(2) OR ARRAY(3)
THEN DO;
  CONTACT$INFO(INDEX). POINTER = CONTACT$INFO(INDEX). POINTER + 1;
  IF CONTACT$INFO(INDEX). POINTER = 15*INDEX + 15
  THEN DO;
    CONTACT$INFO(INDEX). POINTER = 15*INDEX;
    CONTACT$INFO(INDEX). FLAG = 0FFH;
  END;
  IF CONTACT$INFO(INDEX). OS$POINTER <> 0FFH
  THEN DO;
    IF CONTACT$INFO(INDEX). FLAG
    THEN DO;
      TEMP = CONTACT$INFO(INDEX). OS$POINTER;
      COUNT = 1;
      DO WHILE TEMP <> CONTACT$INFO(INDEX). POINTER;
        IF TEMP = (INDEX + 1)*15
        THEN TEMP = INDEX*15;
        ELSE TEMP = TEMP + 1;
        COUNT = COUNT + 1;
      END;
    END;
    ELSE DO;
      COUNT = CONTACT$INFO(INDEX). POINTER -
        CONTACT$INFO(INDEX). OS$POINTER;
    END;
    IF COUNT >= 5
    THEN CONTACT$INFO(INDEX). OS$POINTER = 0FFH;
  END;

```

EXECUTIVE\$CMDS

UPDATE

```

END;
J = CONTACT$INFO(INDEX). POINTER;
CONTACT$POSI(J). TIME(0) = H;
CONTACT$POSI(J). TIME(1) = M;
CONTACT$POSI(J). TIME(2) = S;
END;

DO I = 0 TO LAST(ARRAY);
IF ARRAY(I)
THEN DO;
CALL CRT$PRINT$STRING(. TITLE$3);
CALL SEND$CRLF;
DO CASE I;
DO;
CONTACT$INFO(INDEX). TYPE = GET$TYPE;
END;
DO;
OLD$KIND = CONTACT$INFO(INDEX). KIND;
KIND, CONTACT$INFO(INDEX). KIND = GET$KIND;
SYSTEM. CONTACT$KIND(OLD$KIND) = SYSTEM. CONTACT$KIND(OLD$KIND) - 1;
SYSTEM. CONTACT$KIND(KIND) = SYSTEM. CONTACT$KIND(KIND) + 1;
END;
DO;
CALL GET$COURSE$BRG(1, . CONTACT$POSI(J). BRG);
END;
DO;
CALL GET$RANGE(. CONTACT$POSI(J). RNG);
END;
DO;
CALL GET$COURSE$BRG(0, . CONTACT$POSI(J). CRS);
CONTACT$INFO(INDEX). CRS$FLAG = 1;
END;

```

EXECUTIVE\$CMDS

UPDATE

```
DO;  
    CALL GET$SPEED(. CONTACT$POSI(J). SPD);  
    CONTACT$INFO(INDEX). SPD$FLAG = 1;  
END;  
END; /* END CASE */  
END; /* IF THEN */  
ELSE DO;  
    IF ARRAY(2) OR ARRAY(3)  
    THEN DO;  
        DO CASE I;  
            DO;  
                END;  
            DO;  
                END;  
            DO;  
                END;  
            DO;  
                IF (NOT ARRAY(2)) AND ARRAY(3)  
                THEN DO;  
                    DO K = 0 TO 3;  
                        CONTACT$POSI(J). BRG(K) = CONTACT$POSI(LAST$INFO). BRG(K);  
                    END;  
                END;  
            END;  
            DO;  
                END;  
            DO;  
                IF (NOT ARRAY(3)) AND ARRAY(2)  
                THEN DO;  
                    DO K = 0 TO 3;  
                        CONTACT$POSI(J). RNG(K) = CONTACT$POSI(LAST$INFO). RNG(K);  
                    END;  
                END;  
            END;  
            DO;  
                END;  
            DO;  
                END;  
        END;  
    END;  
END;  
DO;
```

EXECUTIVE\$CMDS

UPDATE

```

DO K = 0 TO 3;
  CONTACT$POSI(J).CRS(K) = CONTACT$POSI(LAST$INFO).CRS(K);
END;
DO;
  DO K = 0 TO 3;
    CONTACT$POSI(J).SPD(K) = CONTACT$POSI(LAST$INFO).SPD(K);
  END;
  /* END CASE */
  /* IF THEN */
  /* ELSE */
  /* END DO */
  IF ARRAY(2) OR ARRAY(3)
  THEN DO;
    CALL CONV$REL$XY(.CONTACT$POSI(J).BRG, .CONTACT$POSI(J).RNG,
      .CONTACT$POSI(J).X, .CONTACT$POSI(J).Y);
    CALL PLASMA$CONTACT(INDEX);
  END;
  IF ARRAY(1)
  THEN CALL DISPLAY$KIND;
  I, OK = 0;
  DO WHILE (OK = 0) AND (I <= LAST(CONTACT$DISPLAY));
    IF CONTACT$DISPLAY(I) = INDEX
    THEN DO;
      OK = 1;
      CALL DISPLAY$CONTACT(I, INDEX);
    END;
    I = I + 1;
  END;
END UPDATE;

```


EXECUTIVE\$CMDS

SWAP\$CONTACTS

```

/*****
*
* SWAP$CONTACTS:
* THIS PROCEDURE IS USED TO SWAP ONE CONTACT BEING DISPLAYED WITH
* ANOTHER WHICH IS IN THE SYSTEM BUT NOT AT THE DISPLAY.
*
*****/
SWAP$CONTACTS: PROCEDURE PUBLIC;
DCL (CONTACT$IN, CONTACT$OUT) ADDRESS,
    STRING (4) BYTE,
    (TEMP, TEMP1, INDEX, I, J) BYTE;
TEMP, TEMP1, J = 0FFH;
DO WHILE TEMP = 0FFH;
    CALL CRT$PRINT$STRING(, TITLE$5);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(, 'ENTER CONTACT TO BE OUT OF DISPLAY:$$');
    CALL SEND$CRLF;
    CONTACT$OUT = GET$DESIG;
    TEMP = CHECK$DESIG(CONTACT$OUT);
    IF TEMP = 0FFH
    THEN DO;
        CALL CRT$PRINT$STRING(, 'CONTACT NOT IN SYSTEM. $$');
        CALL SEND$CRLF;
        CALL CHECK$GO$KEY;
        CALL CLEAR$LOW$SCREEN;
        END;
    ELSE DO;
        DO I = 0 TO LAST(CONTACT$DISPLAY);
            IF CONTACT$DISPLAY(I) = TEMP
            THEN DO;
                J = I;

```

EXECUTIVE#CMDS

SWAP\$CONTACTS

```
I = 6;
END;
END;
IF J = 0FFH
THEN DO;
    TEMP = 0FFH;
    CALL CRT$PRINT$STRING(, 'CONTACT NOT AT DISPLAY. $$$');
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
    END;
END;

END;
DO WHILE TEMP1 = 0FFH;
    CALL CRT$PRINT$STRING(, TITLE$5);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(, 'ENTER CONTACT TO BE IN THE DISPLAY: $$$');
    CALL SEND$CRLF;
    CONTACT$IN = GET$DESIG;
    TEMP1 = CHECK$DESIG (CONTACT$IN);
    IF TEMP1 = 0FFH
    THEN DO;
        CALL CRT$PRINT$STRING(, 'CONTACT NOT IN SYSTEM. $$$');
        CALL SEND$CRLF;
        CALL CHECK$GO$KEY;
        CALL CLEAR$LOW$SCREEN;
        END;
    ELSE DO;
        DO I = 0 TO LAST (CONTACT$DISPLAY);
            IF CONTACT$DISPLAY(I) = TEMP1
            THEN DO;
```

EXECUTIVE\$CMDS

SWAP\$CONTACTS

```
TEMP1 = 0FFH;  
CALL CRT$PRINT$STRING(, ('CONTACT ALREADY DISPLAYED. $$'));  
CALL SEND$CRLF;  
CALL CHECK$GO$KEY;  
CALL CLEAR$LOW$SCREEN;  
I = LAST(CONTACT$DISPLAY) + 2;  
END;  
END;  
END;  
CALL DISPLAY$CONTACT (J, TEMP1);  
CALL CLEAR$LOW$SCREEN;  
END SWAP$CONTACTS;
```

EXECUTIVE#CMDS

TRANSLATE

```

/*****
*
* TRANSLATE:
* THIS PROCEDURE IS USED TO TRANSLATE ALL X,Y VALUES IN THE SYSTEM, BY GI-
* VEN VALUES.
*
* PARAMETERS:
* - A - POINTER TO A FOUR BYTE VECTOR IN WHICH THE CHANGE IN X IS LOCATED.
* - B - POINTER TO A FOUR BYTE VECTOR IN WHICH THE CHANGE IN Y IS LOCATED.
* - TEMP - CAN HAVE TWO VALUES:
*         - 0: DO NOT CHANGE LAST POSITION OF OWN SHIP.
*         - 1: CHANGE ALL X,Y VALUES WITHOUT EXCEPTION.
*
*****/
TRANSLATE: PROCEDURE (A, B, TEMP);
  DCL (A, B) ADDRESS,
        X$DELTA BASED A (4) BYTE,
        Y$DELTA BASED B (4) BYTE,
        (I, J, P, NUM$PTS, TEMP, FLAG) BYTE;
  IF OWN$SHIP$INFO.FLAG THEN NUM$PTS = 29;
  ELSE NUM$PTS = OWN$SHIP$INFO.POINTER;
  DO I = 0 TO NUM$PTS;
    FLAG = 0;
    DO WHILE (<TEMP = 1) OR (<TEMP = 0) AND (I <> OWN$SHIP$INFO.POINTER))
      AND (<FLAG = 0);
      CALL FADD(<OWN$SHIP(I), X, X$DELTA, OWN$SHIP(I), X);
      CALL FADD(<OWN$SHIP(I), Y, Y$DELTA, OWN$SHIP(I), Y);
      FLAG = 1;
    END;
  END;
  DO I = 0 TO 14;

```


EXECUTIVE\$CMDS

TRANSLATE

```
IF CONTACT$INFO(I).DESIG <> 0
THEN DO;
  IF CONTACT$INFO(I).FLAG THEN NUM$PTS = 14;
  ELSE NUM$PTS = CONTACT$INFO(I).POINTER MOD 15;
  DO J = 0 TO NUM$PTS;
    P = I*15 + J;
    CALL FADD(. CONTACT$POSI(P).X, .X$DELTA, . CONTACT$POSI(P).X);
    CALL FADD(. CONTACT$POSI(P).Y, .Y$DELTA, . CONTACT$POSI(P).Y);
  END;
END;
END TRANSLATE;
```

EXECUTIVE\$CMDS

OWN\$SHIP\$UPDATE

```

/*****
*
* OWN$SHIP$UPDATE:
*   THIS PROCEDURE IS USED TO UPDATE THE INFORMATION ABOUT THE OWN SHIP.
*
*****
OWN$SHIP$UPDATE: PROCEDURE PUBLIC;
  DCL ARRAY(4) BYTE,
    OLD$LAT (4) BYTE,
    OLD$LONG (4) BYTE,
    X$DELTA (4) BYTE,
    Y$DELTA (4) BYTE,
    (LAST$INFO, OK, I, J, K, TEMP, H, M, S) BYTE;
  H = HOURS; /* SAVE TIME OF CALL */
  M = MINUTES;
  S = SECONDS;
  /* UPDATE OWN SHIP POSITION */
  CALL MOVE$OWN$SHIP;
  /* GET OWN SHIP VALUES */
  OK = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(. TITLE$4);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(. MSG$1);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(. MSG$4);
    ARRAY(0) = CHECK$YES$NO;
    CALL CRT$PRINT$STRING(. BLANK);
    CALL CRT$PRINT$STRING(. MSG$5);
    ARRAY(1) = CHECK$YES$NO;
    CALL SEND$CRLF;

```

EXECUTIVE\$CMDS

OWN\$SHIP\$UPDATE

```

CALL CRT$PRINT$STRING(. MSG$2);
ARRAY(2) = CHECK$YES$NO;
CALL CRT$PRINT$STRING(. BLANK);
CALL CRT$PRINT$STRING(. MSG$3);
ARRAY(3) = CHECK$YES$NO;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;

TEMP = 0;
DO I = 0 TO LAST(ARRAY);
  IF ARRAY(I) THEN TEMP = 1;
END;

IF TEMP = 0 THEN RETURN; /* NO INPUT IS DESIRED. */
LAST$INFO = OWN$SHIP$INFO. POINTER;
OWN$SHIP$INFO. POINTER = OWN$SHIP$INFO. POINTER + 1;
IF OWN$SHIP$INFO. POINTER = 30
  THEN DO;
  OWN$SHIP$INFO. POINTER = 0;
  OWN$SHIP$INFO. FLAG = 0FFH;
  END;

J = OWN$SHIP$INFO. POINTER;
OWN$SHIP(J). TIME(0) = H;
OWN$SHIP(J). TIME(1) = M;
OWN$SHIP(J). TIME(2) = S;
IF ARRAY(0) OR ARRAY(1)
  THEN DO;
  DO I = 0 TO 3;
    OLD$LAT(I) = OWN$SHIP$INFO. LAT(I);
    OLD$LONG(I) = OWN$SHIP$INFO. LONG(I);
  END;

```

EXECUTIVE\$CMDS

OWN\$SHIP\$UPDATE

```

END;
DO I = 0 TO LAST(ARRAY);
  IF ARRAY(I)
  THEN DO;
    CALL CRT$PRINT$STRING(. TITLE$4);
    CALL SEND$CRLF;
    DO CASE I;
      DO;
        CALL GET$LAT(. OWN$SHIP$INFO. LAT);
      END;
      DO;
        CALL GET$LONG(. OWN$SHIP$INFO. LONG);
      END;
      DO;
        CALL GET$COURSE$BRG(0, . OWN$SHIP(J). CRS);
        TEMP = FP$FORMAT(. OWN$SHIP(J). CRS, . CRS$STRING, 3, 1);
      END;
      DO;
        CALL GET$SPEED(. OWN$SHIP(J). SPD);
        TEMP = FP$FORMAT(. OWN$SHIP(J). SPD, . SPD$STRING, 2, 1);
      END;
      END;
      /* CASE */
      /* IF THEN */
    ELSE DO;
      DO CASE I;
        DO;
          DO K = 0 TO 3;
            OWN$SHIP(J). Y(K) = OWN$SHIP(LAST$INFO). Y(K);
          END;
        END;
      END;
    END;
  END;

```


OWNERSHIP UPDATE

243

EXECUTIVE\$CMDS

OWN\$SHIP\$UPDATE

```
CALL PRINT$LAT$LONG(.LAT$STRING, .LONG$STRING);
CALL CONV$XY(.OLD$LAT, .OLD$LONG, .X$DELTA, .Y$DELTA);
CALL FSUB(.OWN$SHIP(J), X, .X$DELTA, .X$DELTA);
CALL FSUB(.OWN$SHIP(J), Y, .Y$DELTA, .Y$DELTA);
CALL TRANSLATE(.X$DELTA, .Y$DELTA, 0);
CALL CLEAR$STRUCTURES;
CALL SET$WINDOW;
CALL PUT$OS$CENTER;
CALL DRAW$EVERYTHING;
CALL DISPLAY$PLASMA$SCALE;
END;

IF ARRAY(2) THEN CALL PRINT$COURSE(.CRS$STRING);
IF ARRAY(3) THEN CALL PRINT$SPEED(.SPD$STRING);
END OWN$SHIP$UPDATE;
```

EXECUTIVE\$CMDS

ORIGIN

```
/******  
*  
* ORIGIN:  
* THIS PROCEDURE IS USED TO MODIFY THE INFORMATION ABOUT THE  
* COORDINATE GRID ORIGIN.  
*  
*****/  
ORIGIN: PROCEDURE PUBLIC;  
  DCL OLD$LAT (4) BYTE,  
  OLD$LONG (4) BYTE,  
  DELTA$X (4) BYTE,  
  DELTA$Y (4) BYTE,  
  I BYTE;  
  DO I = 0 TO 3;  
    OLD$LAT(I) = SYSTEM.LAT(I);  
    OLD$LONG(I) = SYSTEM.LONG(I);  
  END;  
  CALL CRT$PRINT$STRING(, TITLE$9);  
  CALL SEND$CRLF;  
  CALL GET$LAT(, SYSTEM.LAT);  
  CALL CRT$PRINT$STRING(, TITLE$9);  
  CALL SEND$CRLF;  
  CALL GET$LONG(, SYSTEM.LONG);  
  CALL CONV$XY(, OLD$LAT, , OLD$LONG, , DELTA$X, , DELTA$Y);  
  CALL TRANSLATE(, DELTA$X, , DELTA$Y, 1);  
  CALL CLEAR$STRUCTURES;  
  CALL SET$WINDOW;  
  CALL PUT$OS$CENTER;  
  CALL DRAW$EVERYTHING;  
  CALL DISPLAY$PLASMA$SCALE;  
  END ORIGIN;
```

EXECUTIVE\$CMDS

WIND

```

/***** *****
* WIND:
* THIS PROCEDURE IS USED TO GET INFORMATION ABOUT THE WIND.
*
*****
WIND: PROCEDURE PUBLIC;
  DCL BUFFER(7) BYTE,
  (OK, TEMP) BYTE;
  DCL FP$360 (4) BYTE DATA (00FH, 0FFH, 0B3H, 043H);
  BUFFER(0) = 4;
  BUFFER(1) = 3;
  BUFFER(6) = 0;
  OK = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(, TITLE$A);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(, 'ENTER THE WIND DIRECTION AS REQUESTED: $$$');
    CALL SEND$CRLF;
    TEMP = 0;
    DO WHILE TEMP = 0;
      CALL CRT$PRINT$STRING(, 'DEGREES: $$$');
      CALL PUT$NUMBER$BUFFER(3, , BUFFER(2));
      CALL CRT$WRITE(, ', ');
      CALL PUT$NUMBER$BUFFER(1, , BUFFER(5));
      CALL ASCII$TO$FLOAT(, BUFFER, 7, , SYSTEM.WIND$DIR);
      TEMP = CHECK$FP$VALUE(, SYSTEM.WIND$DIR, , FP$360);
    END;
    CALL SEND$CRLF;
    OK = CHECK$INPUT;
    CALL CLEAR$LOW$SCREEN;
  END;

```


EXECUTIVE\$CMDS

WIND

```
END;
BUFFER(0) = 3;
BUFFER(1) = 2;
BUFFER(5) = 0;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(.TITLE$A);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(.ENTER THE WIND SPEED AS REQUESTED:$$');
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(.KNOTS: $$');
  CALL PUT$NUMBER$BUFFER(2, .BUFFER(2));
  CALL CRT$WRITE(' ');
  CALL PUT$NUMBER$BUFFER(1, .BUFFER(4));
  CALL ASCII$TO$FLOAT(.BUFFER, 6, .SYSTEM.WIND$SPD);
  CALL SEND$CRLF;
  OK = CHECK$INPUT;
  CALL CLEAR$LOW$SCREEN;
END;
END WIND;
```

EXECUTIVE\$CMDS

SCALE

```
/******  
* SCALE:  
* THIS PROCEDURE IS USED TO UPDATE THE GRAPHICS SCALE VALUE.  
*  
*****  
SCALE: PROCEDURE PUBLIC;  
  CALL CRT$PRINT$STRING(, TITLE$B);  
  CALL SEND$CRLF;  
  CALL GET$SCALE(, SYSTEM. SCALE);  
  CALL CLEAR$STRUCTURES;  
  CALL SET$WINDOW;  
  CALL PUT$OS$CENTER;  
  CALL DRAW$EVERYTHING;  
  CALL DISPLAY$PLASMA$SCALE;  
  END SCALE;  
*****
```

EXECUTIVE\$CMDS

GET\$SAFE\$RNG

```

/*****
*
* GET$SAFE$RNG:
* THIS PROCEDURE IS USED TO OBTAIN THE VALUE OF SAFE CPA RANGE USED TO
* WARN THE OPERATOR THAT A CONTACT WILL BE IN COLLISION.
*
*****
GET$SAFE$RNG: PROCEDURE PUBLIC;
DCL BUFFER (7) BYTE,
      (OK, TEMP, TEMP1) BYTE;
DCL LOW$BOUND (4) BYTE DATA (008H, 03CH, 0CAH, 03CH), /* 0.024686827 */
HIGH$BOUND (4) BYTE DATA (000H, 000H, 000H, 03FH), /* 0.5 */
FP$2000 (4) BYTE DATA (0E4H, 02BH, 0FDH, 044H), /* 2025.3716 */
DCL M0 (*) BYTE DATA ('ENTER THE SAFE C.P.A. RANGE AS REQUESTED:$$'),
M1 (*) BYTE DATA ('YARDS: $$');

OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(.TITLE$D);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(.M0);
  CALL SEND$CRLF;
  TEMP, TEMP1 = 0;
  DO WHILE (TEMP = 0) OR (TEMP1 = 0);
    CALL CRT$PRINT$STRING(.M1);
    BUFFER(0), BUFFER(1) = 4;
    BUFFER(6) = 0;
    CALL PUT$NUMBER$BUFFER(4, .BUFFER(2));
    CALL ASCII$TO$FLOAT(.BUFFER, 7, .SAFE$RNG);
    CALL FDIV(.SAFE$RNG, .FP$2000, .SAFE$RNG);
    TEMP = CHECK$FP$VALUE(.SAFE$RNG, .HIGH$BOUND);
  
```

GET\$SAFE\$RNG

EXECUTIVE\$CMDS

```
IF TEMP <> 0
  THEN TEMP1 = CHECK$FP$VALUE(.LOW$BOUND, .SAFE$RNG);
  END;
  CALL SEND$CRLF;
  OK = CHECK$INPUT;
  CALL CLEAR$LOW$SCREEN;
  END;
  END GET$SAFE$RNG;
```


EXECUTIVE\$CMDS

INPUT\$TIME

```

/*****
*
* INPUT$TIME:
* THIS PROCEDURE IS USED TO ALTER ALL VALUES CONCERNING WITH TIME:
*   - TIME ZONE NUMBER.
*   - SYSTEM CLOCK TIME.
*   - TIME BETWEEN UPDATES OF OWN SHIP POSITIONS.
*
* USAGE:
* TYPED PROCEDURE. IF THE TIME BETWEEN UPDATES IS CHANGED (DEFAULT 180 SEC-
* ONDS) THEN THE NEW VALUE IS RETURNED, OTHERWISE A VALUE OF ZERO IS RETUR-
* NED. THE VALUE MUST BE BETWEEN 250 AND 15.
*
*****/
INPUT$TIME: PROCEDURE BYTE PUBLIC;
DCL ARRAY(3) BYTE,
      VALUE ADDRESS,
      (I, OK, TEMP) BYTE;
DCL M0 (*) BYTE DATA ('TIME ZONE NUMBER? (Y/N) $$'),
      M1 (*) BYTE DATA ('SYSTEM CLOCK VALUE? (Y/N) $$'),
      M2 (*) BYTE DATA ('TIME BETWEEN UPDATES? (Y/N) $$'),
      M3 (*) BYTE DATA ('ENTER TIME BETWEEN UPDATES AS REQUESTED:$$'),
      M4 (*) BYTE DATA ('SECONDS: $$'),
      M5 (*) BYTE DATA (' *** BAD FORMAT ***$$');
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(. TITLE$6);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG$1);
  CALL SEND$CRLF;

```

EXECUTIVE\$CMDS

INPUT\$TIME

```
CALL CRT$PRINT$STRING(.M0);
ARRAY(0) = CHECK$YES$NO;
CALL SEND$SPACE(13);
CALL CRT$PRINT$STRING(.M1);
ARRAY(1) = CHECK$YES$NO;
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(.M2);
ARRAY(2) = CHECK$YES$NO;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;

DO I = 0 TO 2;
  IF ARRAY(I)
  THEN DO;
    CALL CRT$PRINT$STRING(.TITLE$6);
    CALL SEND$CRLF;
    DO CASE I;
      DO;
        CALL GET$TIME$ZONE(.SYSTEM.NUM$ZONE);
        CALL PRINT$TIME$ZONE(.SYSTEM.NUM$ZONE);
      END;
      DO;
        CALL INITIATE$TIME;
      END;
      DO;
        OK = 0;
        DO WHILE OK = 0;
          CALL CRT$PRINT$STRING(.M3);
          CALL SEND$CRLF;
          VALUE = 0;
        END;
      END;
    END;
  END;
END;
```

EXECUTIVE#CMDS

INPUT\$TIME

```
DO WHILE (VALUE > 250) OR (VALUE < 15);
  CALL CRT$PRINT$STRING(.M4);
  VALUE = GET$ADDRESS(3);
  IF (VALUE > 250) OR (VALUE < 15)
    THEN DO;
    CALL CRT$PRINT$STRING(.M5);
    CALL SEND$BEL;
    CALL SEND$CR;
    CALL SEND$SUB;
    END;
  ELSE DO;
    CALL CRT$WRITE(17H); /* ERASE TO END OF LINE */
    CALL SEND$CRLF;
    END;
  END;
  OK = CHECK$INPUT;
  CALL CLEAR$LOW$SCREEN;
  END;
END;
END;
/* END CASE */
/* END THEN DO */
/* END DO */
END;
IF ARRAY(2) THEN RETURN LOW(VALUE);
ELSE RETURN 0;
END INPUT$TIME;

END EXECUTIVE#CMDS;
```

CPA\$MODULE

CPA\$MODULE

CPA\$MODULE: DO;
\$NOLIST

\$INCLUDE (:F1:EXTER. SRC)

\$INCLUDE (:F1:EXTER1. SRC)

\$LIST

DCL SAFE\$RNG (4) BYTE EXTERNAL;

CPA\$MODULE

CONV\$CONTACT\$TIME

```

/*****
*
* CONV$CONTACT$TIME:
* THIS PROCEDURE IS USED TO CONVERT A GIVEN CONTACT TIME (IN HOURS,
* MINUTES, AND SECONDS) INTO A FP REPRESENTATION (IN HOURS AND TENTHS
* OF HOURS).
*
* PARAMETERS:
* - S. - POINTER TO A MEMORY LOCATION IN WHICH THE TIME IS LOCATED
*       (IN HOURS, MINUTES, AND SECONDS).
* - T. - POINTER TO A MEMORY LOCATION IN WHICH THE FP VALUE REPRESENTING
*       THE TIME IS DESIRED TO BE PLACED.
*
*****/
CONV$CONTACT$TIME: PROCEDURE (S, T) PUBLIC;
  DCL (S, T) ADDRESS,
        STRING BASED S (3) BYTE,
        TIME$FLOAT BASED T (4) BYTE,
        TEMP (4) BYTE,
        FP$60 (4) BYTE DATA (00H, 00H, 70H, 42H),
        FP$3600 (4) BYTE DATA (00H, 00H, 61H, 45H);
  TEMP(0) = STRING(0);
  TEMP(1), TEMP(2), TEMP(3) = 00H;
  CALL FLTDS (.TEMP, .TIME$FLOAT);
  TEMP(0) = STRING(1);
  TEMP(1), TEMP(2), TEMP(3) = 00H;
  CALL FLTDS (.TEMP, .TEMP);
  CALL FDIY (.TEMP, .FP$60, .TEMP);
  CALL FADD (.TEMP, .TIME$FLOAT, .TIME$FLOAT);
  TEMP(0) = STRING(2);
  TEMP(1), TEMP(2), TEMP(3) = 00H;

```

CONV\$CONTACT\$TIME

CPA\$MODULE

```
CALL FLTDS (.TEMP, .TEMP);  
CALL FDIY (.TEMP, .FP$3600, .TEMP);  
CALL FADD (.TEMP, .TIME$FLOAT, .TIME$FLOAT);  
END CONV$CONTACT$TIME;
```

CPA\$MODULE

CPA\$TIME\$CONV

```

/*****
*
* CPA$TIME$CONV:
* THIS PROCEDURE IS CALLED BY THE "CPA$CALCULATION" PROCEDURE IN
* ORDER TO CONVERT A FP REPRESENTATION OF THE CPA TIME TO A STRING OF
* ASCII CHARACTERS.
*
* PARAMETERS:
* - T - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF
*   THE CPA TIME IS LOCATED.
* - S - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS RE-
*   PRESENTING THE CPA TIME IS DESIRED TO BE PLACED.
*
* *****/
CPA$TIME$CONV: PROCEDURE (T, S);
  DCL (T, S) ADDRESS,
    FP$60 (4) BYTE DATA (00H,00H,70H,42H), /* 60.0 */
    CPA$TIME BASED T (4) BYTE,
    STRING BASED S (4) BYTE,
    <HOURS, MINUTES, TEMP> (4) BYTE,
    J BYTE;
  CALL FIXED (. CPA$TIME, . HOURS);
  DO J = 0 TO 3;
    TEMP(J) = HOURS(J);
  END;
  DO WHILE HOURS(0) >= 24;
    HOURS(0) = HOURS(0) - 24;
  END;
  STRING(0) = HOURS(0) / 10 + 30H;
  STRING(1) = HOURS(0) MOD 10 + 30H;
  CALL FLTDS (. TEMP, . TEMP);

```

CPA\$MODULE

CPA\$TIME\$CONV

```
CALL FSUB (.CPA$TIME, .TEMP, .MINUTES);
CALL FMUL (.MINUTES, .FP$60, .MINUTES);
CALL FIXSD (.MINUTES, .MINUTES);
IF MINUTES(0) >= 60
THEN DO,
    MINUTES(0) = MINUTES(0) - 60;
    HOURS(0) = HOURS(0) + 1;
    IF HOURS(0) >= 24 THEN HOURS(0) = HOURS(0) - 24;
    STRING(0) = HOURS(0) / 10 + 30H;
    STRING(1) = HOURS(0) MOD 10 + 30H;
    END;
    STRING(2) = MINUTES(0) / 10 + 30H;
    STRING(3) = MINUTES(0) MOD 10 + 30H;
END CPA$TIME$CONV;
```


CPA\$MODULE

CONTACT\$CRS\$SPD

```

/*****
*
* CONTACT$CRS$SPD:
* THIS PROCEDURE IS USED TO CALCULATE THE COURSE AND SPEED OF A CONTACT
* GIVEN ITS LAST KNOWN POSITION AT "CONTACT$POSI" STRUCTURE.
* THIS PROCEDURE IS CALLED BY THE "CPA$CALCULATION" PROCEDURE.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION 'N WHICH THE RELATIVE COURSE OF A
*   CONTACT IS LOCATED (IN RADIANS).
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE RELATIVE SPEED OF A
*   CONTACT IS LOCATED (IN KNOTS).
* - S - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS
*   REPRESENTING THE TRUE COURSE AND SPEED OF A CONTACT IS DESIRED
*   TO BE PLACED.
* - INDEX - IT IS THE VALUE WHICH GIVES THE LAST KNOWN POSITION IN
*   THE "CONTACT$POSI" STRUCTURE OF A CONTACT BEING PROCESSED
*   BY THE "CPA$CALCULATION" PROCEDURE.
*
*****
CONTACT$CRS$SPD: PROCEDURE (A, B, S, INDEX);
  DCL (A, B, S) ADDRESS,
  CRS BASED A (4) BYTE,
  SPD BASED B (4) BYTE,
  STRING BASED S (9) BYTE,
  FP$DEG$TO$RAD (4) BYTE DATA (35H,0FAH,8EH,3CH), /* 0.0174532925 */
  (TEMP, TEMP1) (4) BYTE,
  (SIN$CRS, COS$CRS) (4) BYTE,
  (X1, X2, XM, Y1, Y2, YM) (4) BYTE,
  (INDEX, I, J) BYTE;
  J = OWN$SHIP$INFO.POINTER;

```

CPA\$MODULE

CONTACT\$CRS\$SPD

```

DO I = 0 TO 3;
  TEMP(I) = OWN$SHIP(J).CRS(I);
  TEMP1(I) = OWN$SHIP(J).SPD(I);
END;

CALL FMUL (.TEMP, .FP$DEG$TO$RAD, .TEMP);
CALL COS$SIN (.TEMP, .COS$CRS, .SIN$CRS);
CALL FMUL (.TEMP1, .SIN$CRS, .X1);
CALL FMUL (.TEMP1, .COS$CRS, .Y1);
CALL COS$SIN (.CRS, .COS$CRS, .SIN$CRS);
CALL FMUL (.SPD, .SIN$CRS, .X2);
CALL FMUL (.SPD, .COS$CRS, .Y2);
CALL FADD (.X1, .X2, .XM);
CALL FADD (.Y1, .Y2, .YM);
CALL FSQR (.XM, .TEMP1);
CALL FSQR (.YM, .TEMP1);
CALL FADD (.TEMP, .TEMP1, .TEMP); /* TRUE SPEED */
CALL FSQRT (.TEMP, .TEMP); /* TRUE SPEED */
CALL ARCTAN (.YM, .XM, .TEMP1);
CALL FDIV (.TEMP1, .FP$DEG$TO$RAD, .TEMP1); /* TRUE COURSE IN DEGREES */
DO I = 0 TO 3;
  CONTACT$POS(INDEX).CRS(I) = TEMP(I);
  CONTACT$POS(INDEX).SPD(I) = TEMP1(I);
END;

J = INDEX/15;
CONTACT$INFO(J).CRS$FLAG, CONTACT$INFO(J).SPD$FLAG = 6FFH;
J = FP$FORMAT (.TEMP1, .STRING(0), 3, 1);
J = FP$FORMAT (.TEMP, .STRING(4), 2, 1);
END CONTACT$CRS$SPD;

```

CPA\$MODULE

CPA\$CALCULATION

```

/*****
*
* CPA$CALCULATION:
* THIS PROCEDURE IS CALLED BY THE "GET$CPA" PROCEDURE IN ORDER TO
* CALCULATE THE CPA OF A GIVEN CONTACT.
* THE "LEAST SQUARE FIT" METHOD IS USED WITH AT MOST 5 POSITIONS OF
* A GIVEN CONTACT.
*
* PARAMETERS:
* - INDEX1 - IT INDICATES THE FIRST POSITION AT THE "CONTACT$POS1"
* STRUCTURE THAT HAS THE INFORMATION ABOUT THE GIVEN CONTACT.
* - INDEX2 - IT INDICATES THE FIRST POSITION AT THE "CONTACT$POS1"
* STRUCTURE THAT WILL BE USED IN THE "LEAST SQUARE FIT" COMPUTATION.
* - COUNT - IT GIVES THE COUNTING USED TO DETERMINE THE NUMBER OF CONTACT
* POSITIONS TO BE USED IN THE CALCULATION OF THE CPA.
* - S - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS
* REPRESENTING THE CPA INFORMATION IS DESIRED TO BE PLACED.
*
*****/
CPA$CALCULATION: PROCEDURE (INDEX1, INDEX2, COUNT, S);
DCL S ADDRESS,
STRING BASED S (23) BYTE,
CHECK BYTE DATA (04H),
CHECK1 BYTE DATA (02H),
CHECK2 BYTE DATA (05H),
FP$DEG$TO$RAD (4) BYTE DATA (35H,0FAH,0EH,3CH), /* 0.0174532925 */
FP$1 (4) BYTE DATA (00H,00H,80H,3FH), /* 1.0 */
FP$60 (4) BYTE DATA (00H,00H,70H,42H), /* 60.0 */
PI$OVER2 (4) BYTE DATA (00BH,0FH,0C9H,3FH), /* 1.5707963 */
PI$FLOAT (4) BYTE DATA (00BH,0FH,49H,40H), /* 3.141593 */
PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H), /* 4.7123889 */

```

CPA\$MODULE

CPA\$CALCULATION

```

MSG0 (*) BYTE DATA
('SAME CRS & SPD'),
MSG1 (*) BYTE DATA
(' COLLISION'),
MSG2 (*) BYTE DATA
('MOVING AWAY '),
REL$XY (5) STRUCTURE
( X(4) BYTE,
  Y(4) BYTE),
COS$BRG (4) BYTE, SIN$BRG (4) BYTE,
X1 (4) BYTE, Y1 (4) BYTE,
SLOPE (4) BYTE, Y$CUT (4) BYTE,
BIG$Y1 (4) BYTE, BIG$Y2 (4) BYTE,
X$CPA (4) BYTE, Y$CPA (4) BYTE,
CPA$TIME (4) BYTE, CPA$BRG (4) BYTE, CPA$RNG (4) BYTE,
TIME (4) BYTE, TIME1 (4) BYTE,
S0 (4) BYTE, S1 (4) BYTE, S2 (4) BYTE,
T0 (4) BYTE, T1 (4) BYTE,
X$SQUARE (4) BYTE, XY$PROD (4) BYTE,
S1$SQUARE (4) BYTE, ST$PROD (4) BYTE,
NUMERATOR (4) BYTE, DENOMINATOR (4) BYTE,
( REL$CRS, REL$SPD, TEMP$RAD) (4) BYTE,
( INDEX1, INDEX2, COUNT, TEMP, TEMP1, I, J, FLAG, FLAG1, LAST$POINTER) BYTE;

/* LAST$POINTER WILL POINT TO THE LAST POSITION IN CONTACT$POSI */
TEMP = INDEX1 / 15;
LAST$POINTER = 'CONTACT$INFO(TEMP). POINTER;
TEMP = INDEX2;
DO I = 0 TO 3;
  S0(1), S1(1), S2(1), T0(1), T1(1) = 00H;
END;

```


CPA\$MODULE

CPA\$CALCULATION

```

/* COMPUTE PARAMETERS FOR LEAST SQUARE FIT */
S0(0) = COUNT + 1;
CALL FLTDS (.S0, .S0);
DO I = 0 TO COUNT;
  IF TEMP > INDEX1 + 14 THEN TEMP = INDEX1;
  CALL FMUL (.CONTACT$POSI(TEMP), BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
  CALL COS$SIN (.TEMP$RAD, .COS$BRG, .SIN$BRG);
  CALL FMUL (.CONTACT$POSI(TEMP), .RNG, .SIN$BRG, .REL$XY(I), X);
  CALL FMUL (.CONTACT$POSI(TEMP), .RNG, .COS$BRG, .REL$XY(I), Y);
  CALL FADD (.REL$XY(I), X, .S1, .S1);
  CALL FSQR (.REL$XY(I), X, .X$SQUARE);
  CALL FADD (.X$SQUARE, .S2, .S2);
  CALL FADD (.REL$XY(I), Y, .T0, .T0);
  CALL FMUL (.REL$XY(I), Y, .REL$XY(I), X, .XY$PROD);
  CALL FADD (.XY$PROD, .T1, .T1);
  TEMP = TEMP + 1;
END;

CALL FMUL (.S0, .S2, .DENOMINATOR);
CALL FSQR (.S1, .S1$SQUARE);
CALL FSUB (.DENOMINATOR, .S1$SQUARE, .DENOMINATOR);
CALL FMUL (.S0, .T1, .NUMERATOR);
CALL FMUL (.S1, .T0, .ST$PROD);
CALL FSUB (.NUMERATOR, .ST$PROD, .NUMERATOR);
FLAG = FZTST (.DENOMINATOR, .CHECK);
FLAG1 = FZTST (.NUMERATOR, .CHECK);
IF FLAG AND FLAG1 THEN I = 0;
IF FLAG AND (NOT FLAG1) THEN I = 1;
IF (NOT FLAG) AND FLAG1 THEN I = 2;
IF (NOT FLAG) AND (NOT FLAG1) THEN I = 3; /* SLOPE < 0 */
TEMP = LAST$POINTER;
IF TEMP = INDEX1
  /* SLOPE = 0/0 */
  /* SLOPE = X/0 */
  /* SLOPE = 0 */
  /* SLOPE < 0 */

```

CPA\$MODULE

CPA\$CALCULATION

```

THEN TEMP1 = TEMP + 14;
ELSE TEMP1 = LAST$POINTER - 1;
IF (I = 0) AND (FLAG1 := FCMPRC. CONTACT$POSI(TEMP). RNG,
               CONTACT$POSI(TEMP1). RNG, . CHECK2))
THEN I = 1;
IF (I = 1) OR (I = 3)
THEN DO;
  IF FCMPRC. CONTACT$POSI(TEMP). BRG, . CONTACT$POSI(TEMP1). BRG, . CHECK) AND
  FCMPRC. CONTACT$POSI(TEMP). RNG, . CONTACT$POSI(TEMP1). RNG, . CHECK)
  THEN I = 0;
  END;
DO CASE I;
DO;
  /* CONTACT ON SAME CRS & SPD OF OWN SHIP */
  TEMP = LAST$POINTER;
  TEMP1 = OWN$SHIP$INFO. POINTER;
  DO J = 0 TO 3;
    CONTACT$POSI(TEMP). CRS(J) = OWN$SHIP(TEMP1). CRS(J);
    CONTACT$POSI(TEMP). SPD(J) = OWN$SHIP(TEMP1). SPD(J);
  END;
  TEMP1 = FP$FORMAT( CONTACT$POSI(TEMP). CRS, . STRING(0), 3, 1);
  TEMP1 = FP$FORMAT( CONTACT$POSI(TEMP). SPD, . STRING(4), 2, 1);
  TEMP1 = INDEX1 / 15;
  CONTACT$INFO(TEMP1). CRS$FLAG = 0FFH;
  CONTACT$INFO(TEMP1). SPD$FLAG = 0FFH;
  DO J = 7 TO LAST(STRING);
    STRING(J) = MSG0(J - 7);
  END;
END;
DO;
  /* CONTACT RELATIVE COURSE = 000 OR 180 */
  TEMP = LAST$POINTER;
  IF TEMP = INDEX1

```

CPA\$MODULE

CPA\$CALCULATION

```

THEN TEMP1 = TEMP + 14;
ELSE TEMP1 = LAST$POINTER - 1;
FLAG = FCMPC( CONTACT$POSI(TEMP1), RNG, CONTACT$POSI(TEMP), RNG, CHECK1);
CALL FMUL( CONTACT$POSI(INDEX2), BRG, FP$DEG$TO$RAD, TEMP$RAD);
CALL COS$SIN( TEMP$RAD, COS$BRG, SIN$BRG);
CALL FMUL( CONTACT$POSI(INDEX2), RNG, SIN$BRG, X$CPA);
IF X$CPA(3) >= 80H
THEN X$CPA(3) = X$CPA(3) XOR 080H;
/* CONVERT TIME TO FP */
TEMP = LAST$POINTER;
CALL CONV$CONTACT$TIME( CONTACT$POSI(INDEX2), TIME, TIME);
IF CONTACT$POSI(INDEX2), TIME(0) >
CONTACT$POSI(TEMP), TIME(0)
THEN DO;
CONTACT$POSI(TEMP), TIME(0) = CONTACT$POSI(TEMP), TIME(0) + 24;
CALL CONV$CONTACT$TIME( CONTACT$POSI(TEMP), TIME, TIME1);
CONTACT$POSI(TEMP), TIME(0) = CONTACT$POSI(TEMP), TIME(0) - 24;
END;
ELSE CALL CONV$CONTACT$TIME( CONTACT$POSI(TEMP), TIME, TIME1);
/* COMPUTE RELATIVE COURSE */
IF (FLAG1 := FCMPC( REL$XY(COUNT), Y, REL$XY(0), Y, CHECK1))
THEN DO;
DO J = 0 TO 3;
REL$CRS(J) = 00H;
END;
END;
ELSE DO;
DO J = 0 TO 3;
REL$CRS(J) = PI$FLOAT(J);
END;
END;

```

CPA\$MODULE

CPA\$CALCULATION

```

/* COMPUTE RELATIVE SPEED */
CALL FSUB (.TIME1, .TIME, .TIME1)
CALL FSUB (.REL$XY(COUNT), Y, REL$XY(0), Y, Y1)
IF Y1(3) >= 80H THEN Y1(3) = Y1(3) XOR 80H
CALL FDIY (.Y1, .TIME1, REL$SPD)
/* COMPUTE TRUE COURSE AND SPEED */
TEMP = LAST$POINTER
CALL CONTACT$CRS$SPD (.REL$CRS, REL$SPD, .STRING, TEMP)
IF FLAG
THEN DO:
    /* CONTACT CLOSING */
    /* COMPUTE CPA TIME */
    CALL FMUL (.CONTACT$POSI(INDEX2), RNG, COS$BRG, Y$CPA)
    CALL FDIY (.Y$CPA, REL$SPD, CPA$TIME)
    IF CPA$TIME(3) >= 080H
    THEN CPA$TIME(3) = CPA$TIME(3) XOR 080H
    CALL FADD(.CPA$TIME, .TIME, CPA$TIME)
    CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7))
    /* CHECK FOR COLLISION */
    IF (FLAG1 := FCMPR (.SAFE$RNG, X$CPA, .CHECK1))
    THEN DO:
        DO J = 11 TO LAST(STRING)
            STRING(J) = MSG1(J - 11)
        END
        RETURN
    END
    /* COMPUTE CPA BEARING */
    CALL FMUL (.CONTACT$POSI(INDEX2), BRG, FP$DEG$TO$RAD, TEMP$RAD)
    IF (FLAG1 := FCMPR(.PI$FLOAT, TEMP$RAD, .CHECK1))
    THEN DO:
        STRING(11) = '0'
        STRING(12) = '9'

```


CPA\$MODULE

CPA\$CALCULATION

```

STRING(13) = '0'
STRING(14) = '0'
END;
ELSE DO;
  STRING(11) = '2';
  STRING(12) = '7';
  STRING(13) = '0';
  STRING(14) = '0';
END;
/* COMPUTE CPA RANGE */
CALL RANGE$FORMAT (.X$CPA, .STRING(15));
END;
ELSE DO; /* CONTACT MOVING AWAY */
  DO J = 7 TO LAST(STRING);
    STRING(J) = MSG2(J - 7);
  END;
END;
END;
DO; /* CONTACT RELATIVE COURSE = 090 OR 270 */
  TEMP = LAST$POINTER;
  IF TEMP = INDEX1
  THEN TEMP1 = TEMP + 14;
  ELSE TEMP1 = LAST$POINTER - 1;
  FLAG = FCMPR(.CONTACT$POSI(TEMP1).RNG, .CONTACT$POSI(TEMP).RNG,
    .CHECK1);
  CALL FMUL (.CONTACT$POSI(TEMP2).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
  CALL COS$SIN(.TEMP$RAD, .COS$BRG, .SIN$BRG);
  CALL FMUL (.CONTACT$POSI(TEMP2).RNG, .COS$BRG, .Y$CPA);
  IF Y$CPA(3) >= 080H
  THEN Y$CPA(3) = Y$CPA(3) XOR 080H;
  /* CONVERT TIME TO FP */

```

CPA\$MODULE

CPA\$CALCULATION

```

TEMP = LAST$POINTER;
CALL CONV$CONTACT$TIME(. CONTACT$POSI(INDEX2). TIME, . TIME);
IF CONTACT$POSI(INDEX2). TIME(0) >
    CONTACT$POSI(TEMP). TIME(0)
THEN DO;
    CONTACT$POSI(TEMP). TIME(0) = CONTACT$POSI(TEMP). TIME(0) + 24;
    CALL CONV$CONTACT$TIME(. CONTACT$POSI(TEMP). TIME, . TIME1);
    CONTACT$POSI(TEMP). TIME(0) = CONTACT$POSI(TEMP). TIME(0) - 24;
END;
ELSE CALL CONV$CONTACT$TIME(. CONTACT$POSI(TEMP). TIME, . TIME1);
/* COMPUTE RELATIVE COURSE */
IF(FLAG1:= FCMPR (. REL$XY(COUNT). X, . REL$XY(0). X, . CHECK1))
THEN DO;
    DO J = 0 TO 3;
        REL$CRS(J) = PI$OVER2(J);
    END;
END;
ELSE DO;
    DO J = 0 TO 3;
        REL$CRS(J) = PI$3$OVER2(J);
    END;
END;
/* COMPUTE RELATIVE SPEED */
CALL FSUB (. TIME1, . TIME, . TIME1);
CALL FSUB (. REL$XY(COUNT). X, . REL$XY(0). X, . X1);
IF X1(3) >= 80H THEN X1(3) = X1(3) XOR 80H;
CALL FDIV(. X1, . TIME1, . REL$SPD);
/* COMPUTE TRUE COURSE AND SPEED */
TEMP = LAST$POINTER;
CALL CONTACT$CRS$SPD (. REL$CRS, . REL$SPD, . STRING, TEMP);
IF FLAG

```

CPA\$MODULE

CPA\$CALCULATION

```

THEN DO;
    /* CONTACT CLOSING */
    /* COMPUTE CPA TIME */
    CALL FMUL(.CONTACT$POS1(INDEX2).RNG, .SIN$BRG, .X$CPA);
    CALL FDIV(.X$CPA, .REL$SPD, .CPA$TIME);
    IF CPA$TIME(3) >= 080H
        THEN CPA$TIME(3) = CPA$TIME(3) XOR 080H;
    CALL FADD(.CPA$TIME, .TIME, .CPA$TIME);
    CALL CPA$TIME$CONV (.CPA$TIME, .STRING(7));
    /* CHECK FOR COLLISION */
    IF (FLAG1 := FCMPR (.SAFE$RNG, .Y$CPA, .CHECK1))
    THEN DO;
        DO J = 11 TO LAST(STRING);
            STRING(J) = MSG1(J - 11);
        END;
        RETURN;
    END;
    /* COMPUTE CPA BEARING */
    CALL FMUL(.CONTACT$POS1(INDEX2).BRG, .FP$DEG$TO$RAD, .TEMP$RAD);
    IF (FLAG := FCMPR(.PI$3$OVER2, .TEMP$RAD, .CHECK1))
    AND (FLAG := FCMPR(.TEMP$RAD, .PI$OVER2, .CHECK1))
    THEN DO;
        STRING(11) = '1';
        STRING(12) = '8';
        STRING(13) = '0';
        STRING(14) = '0';
    END;
    ELSE DO;
        STRING(11) = '0';
        STRING(12) = '0';
        STRING(13) = '0';
        STRING(14) = '0';
    END;

```

CPA\$MODULE

CPA\$CALCULATION

```

END;
/* COMPUTE CPA RANGE */
CALL RANGE$FORMAT (.Y$CPA, .STRING(15));
END;
ELSE DO; /* CONTACT MOVING AWAY */
DO J = 7 TO LAST(STRING);
STRING(J) = MSG2(J - 7);
END;
END;

END;
/* RELATIVE MOVE - GENERAL CASE */
DO;
TEMP = LAST$POINTER;
IF TEMP = INDEX1
THEN TEMP1 = TEMP + 14;
ELSE TEMP1 = LAST$POINTER - 1;
FLAG = FCMPR( CONTACT$POSI(TEMP1), RNG, CONTACT$POSI(TEMP), RNG,
CHECK1);
/* PERFORM LEAST SQUARE FIT FOR LAST POSITIONS */
CALL FDIV (.NUMERATOR, .DENOMINATOR, .SLOPE);
/* COMPUTE SLOPE */
/* COMPUTE CUT AT Y AXIS */
CALL FMUL (.S2, .T0, .NUMERATOR);
CALL FMUL (.S1, .T1, .ST$PROD);
CALL FSUB (.NUMERATOR, .ST$PROD, .NUMERATOR);
CALL FDIV (.NUMERATOR, .DENOMINATOR, .Y$CUT);
/* COMPUTE RELATIVE COURSE */
/* PREPARE TO COMPUTE RELATIVE SPEED */
CALL FMUL (.SLOPE, .REL$XY(0), X, .BIG$Y1);
CALL FADD (.BIG$Y1, .Y$CUT, .BIG$Y1);
CALL FMUL (.SLOPE, .REL$XY(COUNT), X, .BIG$Y2);
CALL FADD (.BIG$Y2, .Y$CUT, .BIG$Y2);

```


CPA\$MODULE

CPA\$CALCULATION

```

CALL FSUB(.BIG$Y2, .BIG$Y1, .NUMERATOR);
CALL FSUB(.REL$XY(COUNT), X, .REL$XY(0), X, .DENOMINATOR);
CALL ARC$TAN(.NUMERATOR, .DENOMINATOR, .REL$CRS);
CALL FSQR(.NUMERATOR, .NUMERATOR);
CALL FSQR(.DENOMINATOR, .DENOMINATOR);
CALL FADD(.NUMERATOR, .DENOMINATOR, .NUMERATOR);
CALL FSQRT(.NUMERATOR, .NUMERATOR);
/* CONVERT TIME TO FP */
TEMP = LAST$POINTER;
CALL CONV$CONTACT$TIME(.CONTACT$POSI(INDEX2), TIME, .TIME);
IF CONTACT$POSI(INDEX2).TIME(0) >
CONTACT$POSI(TEMP).TIME(0)
THEN DO;
    CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) + 24;
    CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP), TIME, .TIME1);
    CONTACT$POSI(TEMP).TIME(0) = CONTACT$POSI(TEMP).TIME(0) - 24;
END;
ELSE DO;
    CALL CONV$CONTACT$TIME(.CONTACT$POSI(TEMP), TIME, .TIME1);
END;
CALL FSUB(.TIME1, .TIME, .TIME1);
/* COMPUTE RELATIVE SPEED */
CALL FDIV(.NUMERATOR, .TIME1, .REL$SPD);
/* COMPUTE TRUE COURSE AND SPEED */
TEMP = LAST$POINTER;
CALL CONTACT$CRS$SPD (.REL$CRS, .REL$SPD, .STRING, TEMP);
IF FLAG THEN DO;
    /* CONTACT CLOSING */
    /* COMPUTE CPA */
    CALL FMUL(.SLOPE, .REL$XY(0), X, .NUMERATOR);
    CALL FSUB(.NUMERATOR, .BIG$Y1, .NUMERATOR);

```

CPA\$MODULE

CPA\$CALCULATION

```

CALL FMUL<.SLOPE, .NUMERATOR, .NUMERATOR>;
CALL FSQR<.SLOPE, .DENOMINATOR>;
CALL FADD<.DENOMINATOR, .FP$1, .DENOMINATOR>;
/* COMPUTE X$CPA */
CALL FDIV<.NUMERATOR, .DENOMINATOR, .X$CPA>;
/* COMPUTE Y$CPA */
CALL FMUL<.SLOPE, .REL$XY<0>().X, .NUMERATOR>;
CALL FSUB<.BIG$Y1, .NUMERATOR, .NUMERATOR>;
CALL FDIV<.NUMERATOR, .DENOMINATOR, .Y$CPA>;
/* COMPUTE CPA TIME */
CALL FSUB<.X$CPA, .REL$XY<0>().X, .NUMERATOR>;
CALL FSQR<.NUMERATOR, .NUMERATOR>;
CALL FSUB<.Y$CPA, .BIG$Y1, .DENOMINATOR>;
CALL FSQR<.DENOMINATOR, .DENOMINATOR>;
CALL FADD<.NUMERATOR, .DENOMINATOR, .NUMERATOR>;
CALL FSQRT<.NUMERATOR, .NUMERATOR>;
CALL FDIV<.NUMERATOR, .REL$SPD, .CPA$TIME>;
CALL FADD<.TIME, .CPA$TIME, .CPA$TIME>;
CALL CPA$TIME$CONV<.CPA$TIME, .STRING<7>>>;
/* COMPUTE CPA RANGE */
CALL FSQR<.X$CPA, .NUMERATOR>;
CALL FSQR<.Y$CPA, .DENOMINATOR>;
CALL FADD<.NUMERATOR, .DENOMINATOR, .NUMERATOR>;
CALL FSQRT<.NUMERATOR, .CPA$RNG>;
/* CHECK FOR COLLISION */
IF <FLAG1 := FCMPR<.SAFE$RNG, .CPA$RNG, .CHECK1>>
THEN DO;
DO J = 11 TO LAST<STRING>;
STRING<J> = MSG1<J - 11>;
END;
RETURN;

```

CPA\$MODULE

CPA\$CALCULATION

```

END;
/* COMPUTE CPA BEARING */
CALL ARCTAN(.Y$CPA, .X$CPA, .CPA$BRG);
CALL CONV$RAD$MIN(.CPA$BRG, .CPA$BRG);
CALL FDIV(.CPA$BRG, .FP$60, .CPA$BRG);
TEMP = FP$FORMAT (.CPA$BRG, .STRING(11), 3, 1);
CALL RANGE$FORMAT(.CPA$BRG, .STRING(15));
END;
ELSE DO;
/* CONTACT MOVING AWAY */
DO J = 7 TO LAST(STRING);
STRING(J) = MSG2(J - 7);
END;
END;
END;
/* END CASE */
END CPA$CALCULATION;

```

CPA\$MODULE

GET\$CPA

```

/*****
*
* GET$CPA:
*   THIS PROCEDURE IS USED TO FIND THE CPA INFORMATION ABOUT A CONTACT.
*
* PARAMETERS:
*   - INDEX - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE
*     CONTACT$INFO STRUCTURE.
*   - A - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF CHARACTERS
*     REPRESENTING THE CPA INFORMATION IS DESIRED TO BE PLACED.
*
* USAGE:
*   TYPED PROCEDURE. RETURNS A VALUE OF 0 IF NO CPA CAN BE DETERMINED AT
*   THE MOMENT. OTHERWISE, A VALUE OF 1 IS RETURNED.
*
*****/
GET$CPA: PROCEDURE (INDEX, A) BYTE PUBLIC;
  DCL A ADDRESS;
  STRING BASED A (23) BYTE,
    (INDEX, INDEX1, COUNT, TEMP, P1, P2, I) BYTE;
  DO I = 0 TO LAST(STRING);
    STRING(I) = 020H; /* PUT BLANKS IN WORK BUFFER */
  END;
  INDEX1 = 15*INDEX;
  IF (CONTACT$INFO(INDEX). POINTER < INDEX1 + 1) AND
    (NOT CONTACT$INFO(INDEX). FLAG)
  THEN RETURN 0;
  P1 = CONTACT$INFO(INDEX). POINTER;
  P2 = CONTACT$INFO(INDEX). OS$POINTER;
  IF CONTACT$INFO(INDEX). FLAG
  THEN DO;

```


CPA\$MODULE

GET\$CPA

```
IF P2 = 0FFH
THEN DO;
  COUNT = 4;
  IF P1 < INDEX1 + 4
  THEN TEMP = P1 + 11;
  ELSE TEMP = P1 - 4;
END;
ELSE DO;
  COUNT = 0;
  TEMP = P2 + 1;
  IF TEMP = (INDEX1 + 15)
  THEN TEMP = INDEX1;
  DO WHILE TEMP <> P1;
    IF TEMP = (INDEX1 + 15)
    THEN TEMP = INDEX1;
    ELSE TEMP = TEMP + 1;
    COUNT = COUNT + 1;
  END;
  IF COUNT = 0
  THEN RETURN 0;
  TEMP = P2 + 1;
  IF TEMP = (INDEX1 + 15)
  THEN TEMP = INDEX1;
END;
ELSE DO;
  IF P2 = 0FFH
  THEN DO;
    COUNT = P1 - INDEX1;
    IF COUNT > 4 THEN COUNT = 4;
    TEMP = P1 - COUNT;
  
```

CPA\$MODULE

GET\$CPA

```
END;  
ELSE DO;  
  IF P1 = P2 THEN RETURN 0;  
  COUNT = P1 - P2 - 1;  
  TEMP = P2 + 1;  
  IF COUNT = 0 THEN RETURN 0;  
END;  
END;  
CALL CPA$CALCULATION(INDEX1, TEMP, COUNT, .STRING);  
RETURN 1;  
END GET$CPA;
```

END CPA\$MODULE;

DISPLAY\$CMDS

DISPLAY\$CMDS

DISPLAY\$CMDS: DO;

\$NOLIST

\$INCLUDE (:F1:EXTER.SRC)

\$INCLUDE (:F1:EXTER1.SRC)

\$LIST

GET\$CPA:

PROCEDURE (A,B) BYTE EXTERNAL;
DCL A BYTE, B ADDRESS; END;

CONV\$CONTACT\$TIME:

PROCEDURE (A, B) EXTERNAL;
DCL (A, B) ADDRESS; END;

DCL SAFE\$RNG (4) BYTE EXTERNAL;

DCL

TITLE\$0 (*) BYTE DATA
(

TITLE\$1 (*) BYTE DATA
(

TITLE\$2 (*) BYTE DATA
(

TITLE\$3 (*) BYTE DATA

COORDINATE GRID ORIGIN\$\$'),

GRAPHICS SCALE\$\$'),

OWN SHIP INFORMATION\$\$'),

DISPLAY\$CMDS

DISPLAY\$CMDS

```

('
TITLE$4 (*) BYTE DATA
('
TITLE$5 (*) BYTE DATA
('
TITLE$6 (*) BYTE DATA
('
TITLE$7 (*) BYTE DATA
('

CONTACT INFORMATION$$'),
SYSTEM INFORMATION. $$'),
CURRENT SAFE C. P. A. RANGE $$'),
WIND INFORMATION$$'),
CURRENT TIME BETWEEN UPDATES. $$');

```

DCL

```

MSG$00 (*) BYTE DATA ('POSITIONAL DATA:$$'),
MSG$01 (*) BYTE DATA ('TACTICAL DATA AT $$'),
MSG$02 (*) BYTE DATA ('C. P. A. DATA:$$'),
MSG$03 (*) BYTE DATA ('GENERAL DATA:$$');

```

DCL

```

PLUS$SIGN LIT '02BH',
MINUS$SIGN LIT '02DH',
COLON LIT '03AH',
POINT LIT '02EH',
BLANK LIT '020H';

```


DISPLAY\$CMDS

CONV\$LAT\$LONG

```

/*****
*
* CONV$LAT$LONG:
* THIS PROCEDURE IS USED TO CONVERT GIVEN 'X,Y' COORDINATES, INTO LATITUDE
* AND LONGITUDE VALUES.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF 'X'
*   IS LOCATED.
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF 'Y'
*   IS LOCATED.
* - C - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF THE LATITUDE, IN
*   MINUTES, IS DESIRED TO BE PLACED.
* - D - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF THE LONGITUDE, IN
*   MINUTES, IS DESIRED TO BE PLACED.
*
*****
CONV$LAT$LONG: PROCEDURE (A,B,C,D) PUBLIC
    DCL (A,B,C,D) ADDRESS,
        X BASED A (4) BYTE,
        Y BASED B (4) BYTE,
        LAT BASED C (4) BYTE,
        LONG BASED D (4) BYTE,
        MEAN$LAT (4) BYTE,
        COS$MEAN$LAT (4) BYTE,
        SIN$MEAN$LAT (4) BYTE,
        FP$2 (4) BYTE DATA (00H, 00H, 00H, 00H, 40H);
    CALL FADD(C,Y, .SYSTEM.LAT, .LAT);
    CALL FADD(C,SYSTEM.LAT, .LAT, .MEAN$LAT);
    CALL FDIV(C,MEAN$LAT, .FP$2, .MEAN$LAT);
    CALL CONV$MIN$RAD(C,MEAN$LAT, .MEAN$LAT);

```

DISPLAY\$CMDS

CONV\$LAT\$LONG

```
CALL COS$SIN( MEAN$LAT, .COS$MEAN$LAT, .SIN$MEAN$LAT);  
CALL FDI( X, .COS$MEAN$LAT, .LONG);  
CALL FADD( .LONG, .SYSTEM LONG, .LONG);  
END CONV$LAT$LONG;
```

DISPLAY\$CMDS

DISPLAY\$DESIG

```

/*****
*
* DISPLAY$DESIG:
* THIS PROCEDURE IS USED TO DISPLAY THE DESIG CHARACTERS.
*
* PARAMETERS:
* - DESIG - ADDRESS VALUE REPRESENTING THE 'HASHED' VALUE OF THE DESIGNATION
* DESIRED TO BE DISPLAYED.
*
*****/
DISPLAY$DESIG: PROCEDURE(DESIG) PUBLIC;
  DCL DESIG ADDRESS,
    CHAR(4) BYTE;
  DCL D (10) BYTE DATA ('DESIG: $$');
  CALL CRT$PRINT$STRING(D);
  CALL DE$HASH(DESIG, .CHAR);
  CHAR(2), CHAR(3) = '$';
  CALL CRT$PRINT$STRING(.CHAR);
  END DISPLAY$DESIG;

```

DISPLAY\$CMDS

DISPLAY\$TYPE

```

/*****
*
* DISPLAY$TYPE:
* THIS PROCEDURE IS USED TO DISPLAY THE TYPE OF A CONTACT.
*
* PARAMETERS:
* - A - POINTER TO A BYTE VALUE REPRESENTING THE TYPE TO BE DISPLAYED.
*
*****/
DISPLAY$TYPE: PROCEDURE (A) PUBLIC;
DCL A ADDRESS,
      TYPE BASED A BYTE;
DCL S (*) BYTE DATA ('SURFACE' $$$),
      SS (*) BYTE DATA ('SUB-SURFACE$$$'),
      T (10) BYTE DATA ('TYPE: $$$');
CALL CRT$PRINT$STRING(.T);
IF TYPE = 0
  THEN CALL CRT$PRINT$STRING(.S);
ELSE CALL CRT$PRINT$STRING(.SS);
END DISPLAY$TYPE;

```


DISPLAY\$CMDS

DISPLAY\$CLASS

```
/******  
* DISPLAY$CLASS:  
* THIS PROCEDURE IS USED TO DISPLAY THE CLASS OF A CONTACT.  
*  
* PARAMETERS:  
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE CLASS TO BE DISPLAYED IS  
* LOCATED.  
*  
*****  
DISPLAY$CLASS: PROCEDURE (A) PUBLIC;  
  DCL A ADDRESS;  
  CLASS BASED A BYTE;  
  DCL FRI (*) BYTE DATA (' FRIEND$$'),  
  HOS (*) BYTE DATA (' HOSTILE$$'),  
  UNK (*) BYTE DATA (' UNKNOWN$$'),  
  C (10) BYTE DATA (' CLASS: $$');  
  
  CALL CRT$PRINT$STRING(C);  
  DO CASE CLASS;  
    CALL CRT$PRINT$STRING(C FRI);  
    CALL CRT$PRINT$STRING(C HOS);  
    CALL CRT$PRINT$STRING(C UNK);  
  END;  
  END DISPLAY$CLASS;
```

DISPLAY\$CMDS

DISPLAY\$LAT\$LONG

```

/*****
*
* DISPLAY$LAT$LONG:
* THIS PROCEDURE IS USED TO DISPLAY THE VALUES OF LATITUDE AND LONGITUDE.
*
* PARAMETERS:
* - KIND. - DENOTES LATITUDE IF 0, OTHERWISE DENOTES LONGITUDE.
* - A. - POINTER TO THE FP REPRESENTATION OF LAT/LONG.
*
*****/
DISPLAY$LAT$LONG: PROCEDURE (KIND, A) PUBLIC;
DCL A ADDRESS,
    VALUE BASED A (4) BYTE,
    CHAR (9) BYTE,
    (KIND, 1) BYTE;
DCL LAT (14) BYTE DATA ('LATITUDE:  '),
    LONG (14) BYTE DATA ('LONGITUDE:  '),
    NORTH (8) BYTE DATA (' NORTH'),
    SOUTH (8) BYTE DATA (' SOUTH'),
    EAST (8) BYTE DATA (' EAST '),
    WEST (8) BYTE DATA (' WEST ');
IF KIND = 0
THEN DO;
    CALL CRT$PRINT$STRING(.LAT);
    CALL LAT$LONG$FORMAT(A, .CHAR, 0);
END;
ELSE DO;
    CALL CRT$PRINT$STRING(.LONG);
    CALL LAT$LONG$FORMAT(A, .CHAR, 1);
END;

```

DISPLAY\$CMDS

DISPLAY\$LAT\$LONG

```
DO I = 0 TO 5;
  IF I = 3 THEN CALL CRT$WRITE(COLON);
  IF I = 5 THEN CALL CRT$WRITE(POINT);
  CALL CRT$WRITE(CHAR(I));
  END;
  IF KIND = 0
  THEN DO;
    IF CHAR(6) = 'N'
    THEN CALL CRT$PRINT$STRING(. NORTH);
    ELSE CALL CRT$PRINT$STRING(. SOUTH);
    END;
  ELSE DO;
    IF CHAR(6) = 'E'
    THEN CALL CRT$PRINT$STRING(. EAST);
    ELSE CALL CRT$PRINT$STRING(. WEST);
    END;
  END DISPLAY$LAT$LONG;
```

DISPLAY\$CMDS

DISPLAY\$XY

```

/*****
*
* DISPLAY$XY:
* THIS PROCEDURE IS USED TO DISPLAY VALUES OF 'X,Y' COORDINATES.
*
* PARAMETERS:
* - TYPE - IF 0 'X' WILL BE DISPLAYED, OTHERWISE 'Y' WILL BE DISPLAYED.
* - A - POINTER TO A FP REPRESENTATION OF THE X/Y VALUE.
*
*****/
DISPLAY$XY: PROCEDURE (TYPE, A) PUBLIC;
  DCL A ADDRESS,
        CHAR (14) BYTE,
        (TYPE, TEMP, I) BYTE;
  DCL X (8) BYTE DATA ('X:  $$$'),
        Y (8) BYTE DATA ('Y:  $$$');

  IF TYPE = 0
    THEN CALL CRT$PRINT$STRING(X);
  ELSE CALL CRT$PRINT$STRING(Y);
  TEMP = FP$FORMAT(A, .CHAR, 10, 2);
  IF TEMP = 0
    THEN CALL CRT$WRITE(PLUS$SIGN);
  ELSE CALL CRT$WRITE(MINUS$SIGN);
  DO I = 0 TO 11;
    IF I = 10 THEN CALL CRT$WRITE(POINT);
    CALL CRT$WRITE(CHAR(I));
  END;
END DISPLAY$XY;

```


DISPLAY\$CMDS

DISPLAY\$CRS\$BRG

```

/*****
*
* DISPLAY$CRS$BRG:
* THIS PROCEDURE IS USED TO DISPLAY THE VALUES OF COURSE AND BEARING.
*
* PARAMETERS:
* - KIND. - IF 0 THE VALUE OF COURSE WILL BE DISPLAYED, OTHERWISE THE VALUE
* OF BEARING WILL BE DISPLAYED.
* - A. - POINTER TO THE FP REPRESENTATION OF COURSE/BEARING.
*
*****/
DISPLAY$CRS$BRG: PROCEDURE (KIND, A) PUBLIC;
  DCL A ADDRESS,
        CHAR (6) BYTE,
        (I, TEMP, KIND) BYTE;
  DCL CRS (12) BYTE DATA ('COURSE:  $'),
        BRG (12) BYTE DATA ('BEARING:  $');

  IF KIND = 0
    THEN CALL CRT$PRINT$STRING(. CRS);
    ELSE CALL CRT$PRINT$STRING(. BRG);
  TEMP = FP$FORMAT(A, . CHAR, 3, 1);
  DO I = 0 TO 3;
    IF I = 3 THEN CALL CRT$WRITE(. POINT);
    CALL CRT$WRITE(. CHAR(I));
  END;
END DISPLAY$CRS$BRG;

```

AD-A059 603

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 17/7

A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOT--ETC(U)

JUN 78 A L GONCALVES, J E CUBA BRAVO

UNCLASSIFIED

NL

4 OF 6
AD
A059603



IFIED

4 OF 6

AD
A 0 59603



DISPLAY\$CMDS

DISPLAY\$SPD

```

/*****
*
* DISPLAY$SPD:
* THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF SPEED.
*
* PARAMETERS:
* - A - POINTER TO THE FP REPRESENTATION OF THE VALUE OF SPEED.
*
*****/
DISPLAY$SPD: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        CHAR (5) BYTE,
        (TEMP, I) BYTE;
  DCL SPD (12) BYTE DATA ('SPEED:  $$$');

  CALL CRT$PRINT$STRING(.SPD);
  TEMP = FP$FORMAT(A, .CHAR, 2, 1);
  DO I = 0 TO 2;
    IF I = 2 THEN CALL CRT$WRITE(POINT);
    CALL CRT$WRITE(CHAR(I));
  END;
END DISPLAY$SPD;

```


DISPLAY\$CMDS

DISPLAY\$RANGE

```

/*****
*
* DISPLAY$RANGE:
* THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF RANGE.
*
* PARAMETERS:
* - A - POINTER TO THE FP REPRESENTATION OF RANGE.
*
*****/
DISPLAY$RANGE: PROCEDURE (A) PUBLIC;
DCL A ADDRESS,
CHAR (8) BYTE,
I BYTE;
DCL MLS (8) BYTE DATA (' MILES$$'),
YDS (8) BYTE DATA (' YARDS$$'),
RNG (12) BYTE DATA ('RANGE: $$');

CALL CRT$PRINT$STRING(.RNG);
CALL RANGE$FORMAT(A, .CHAR);
DO I = 0 TO 4;
CALL CRT$WRITE(CHAR(I));
END;
IF CHAR(5) = 'M'
THEN CALL CRT$PRINT$STRING(.MLS);
ELSE CALL CRT$PRINT$STRING(.YDS);
END DISPLAY$RANGE;

```

DISPLAY\$CMDS

DISPLAY\$TIME

```

/*****
*
* DISPLAY$TIME:
*   THIS PROCEDURE IS USED TO DISPLAY THE VALUE OF THE TIME.
*
* PARAMETERS:
*   - A - POINTER TO A 3 BYTE VECTOR CONTAINING THE TIME VALUE.
*
*****/
DISPLAY$TIME: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        DIGIT BASED A BYTE,
        CHAR BYTE;

  DCL TIME (12) BYTE DATA ('TIME:  $$');

  CALL CRT$PRINT$STRING(. TIME);
  CHAR = DIGIT/10 + 30H;
  CALL CRT$WRITE(CHAR);
  CHAR = DIGIT MOD 10 + 30H;
  CALL CRT$WRITE(CHAR);
  A = A + 1;
  CALL CRT$WRITE(COLON);
  CHAR = DIGIT/10 + 30H;
  CALL CRT$WRITE(CHAR);
  CHAR = DIGIT MOD 10 + 30H;
  CALL CRT$WRITE(CHAR);
  A = A + 1;
  CALL CRT$WRITE(COLON);
  CHAR = DIGIT/10 + 30H;
  CALL CRT$WRITE(CHAR);
  CHAR = DIGIT MOD 10 + 30H;

```

DISPLAY\$CMDS

DISPLAY\$TIME

CALL CRT\$WRITE(CHAR);
END DISPLAY\$TIME;

DISPLAY\$CMDS

DISPLAY\$ORIGIN

```
/*
*
* DISPLAY$ORIGIN:
* THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE COORDINATE GRID
* ORIGIN.
*
* *****/
DISPLAY$ORIGIN: PROCEDURE PUBLIC;
DCL MSG (*) BYTE DATA
      ('THE COORDINATE GRID ORIGIN VALUES ARE:$$');

      CALL CRT$PRINT$STRING(. TITLE$0);
      CALL SEND$CRLF;
      CALL CRT$PRINT$STRING(. MSG);
      CALL SEND$CRLF;
      CALL DISPLAY$LAT$LONG(0, . SYSTEM. LAT);
      CALL SEND$CRLF;
      CALL DISPLAY$LAT$LONG(1, . SYSTEM. LONG);
      CALL SEND$CRLF;
      CALL SEND$CRLF;
      CALL CHECK$GO$KEY;
      CALL CLEAR$LOW$SCREEN;
      END DISPLAY$ORIGIN;
```


DISPLAY\$CMDS

DISPLAY\$SCALE

```

/*****
*
* DISPLAY$SCALE:
* THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE SCALE BEING USED
* IN THE GRAPHICS DISPLAY.
*
*****/
DISPLAY$SCALE: PROCEDURE PUBLIC;
  DCL CHAR (6) BYTE,
    (1, TEMP) BYTE;
  DCL MSG (*) BYTE DATA
    ('THE VALUE OF THE GRAPHICS SCALE IS: $$');

  CALL CRT$PRINT$STRING(, TITLE$1);
  CALL SEND$CRLF;
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(, MSG);
  TEMP = FP$FORMAT(, SYSTEM.SCALE, , CHAR, 2, 2);
  DO I = 0 TO 3;
    IF I = 2 THEN CALL CRT$WRITE(POINT);
    CALL CRT$WRITE(CHAR(I));
  END;
  CALL SEND$CRLF;
  CALL SEND$CRLF;
  CALL CHECK$GO$KEY;
  CALL CLEAR$LOW$SCREEN;
  END DISPLAY$SCALE;

```

DISPLAY\$CMDS

DISPLAY\$OWN\$SHIP

```

/*****
*
* DISPLAY$OWN$SHIP:
* THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE OWN SHIP.
*
*****/
DISPLAY$OWN$SHIP: PROCEDURE PUBLIC;
DCL POINTER BYTE;
CALL CRT$PRINT$STRING(, TITLE$2);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(, MSG$00);
CALL SEND$CRLF;
CALL DISPLAY$LAT$LONG(0, .OWN$SHIP$INFO.LAT);
CALL SEND$SPACE(12);
CALL DISPLAY$LAT$LONG(1, .OWN$SHIP$INFO.LONG);
CALL SEND$CRLF;
POINTER = OWN$SHIP$INFO.POINTER;
CALL DISPLAY$XY(0, .OWN$SHIP(POINTER).X);
CALL SEND$SPACE(18);
CALL DISPLAY$XY(1, .OWN$SHIP(POINTER).Y);
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
CALL CRT$PRINT$STRING(, TITLE$2);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(, MSG$01);
CALL DISPLAY$TIME(, .OWN$SHIP(POINTER).TIME);
CALL SEND$CRLF;
CALL DISPLAY$CRS$BRG(0, .OWN$SHIP(POINTER).CRS);
CALL SEND$SPACE(20);

```

DISPLAY\$OWN\$SHIP

DISPLAY\$CMDS

```
CALL DISPLAY$SPD(.OWN$SHIP<POINTER>).SPD);  
CALL SEND$CRLF;  
CALL SEND$CRLF;  
CALL CHECK$GO$KEY;  
CALL CLEAR$LOW$SCREEN;  
END DISPLAY$OWN$SHIP;
```

DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```
/******  
* DISPLAY$CONTACT$INFO:  
* THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT ANY CONTACT BEING  
* HANDLED BY THE SYSTEM.  
*  
*****/  
DISPLAY$CONTACT$INFO: PROCEDURE PUBLIC;  
  DCL DESIG ADDRESS,  
    LAT (4) BYTE,  
    LONG (4) BYTE,  
    TIME$1 (4) BYTE, TIME$2 (4) BYTE,  
    X$DELTA (4) BYTE, Y$DELTA (4) BYTE,  
    COS$DIST (4) BYTE, SIN$DIST (4) BYTE,  
    BRG (4) BYTE, RNG (4) BYTE,  
    T (3) BYTE, DISTANCE (4) BYTE,  
    STRING (23) BYTE,  
    (TEMP, OK, INDEX, I, POINTER) BYTE;  
  DCL DEG$TO$RAD (4) BYTE DATA (035H, 0FAH, 08EH, 03CH); /* 0.0174532925 */  
  
  DCL MSG0 (*) BYTE DATA  
    ('NO COURSE INFORMATION AVAILABLE. $$'),  
    MSG1 (*) BYTE DATA  
    ('NO SPEED INFORMATION AVAILABLE. $$'),  
    MSG2 (*) BYTE DATA  
    ('NO CPA INFORMATION AVAILABLE. $$'),  
    MSG3 (*) BYTE DATA  
    ('ENTER CONTACT DESIG AS REQUESTED: $$'),  
    MSG4 (*) BYTE DATA  
    ('DESIG NOT IN USE. $$'),  
    MSG5 (*) BYTE DATA
```


DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```
      ('CURRENT NUMBER OF POSITIONS: $$'),
MSG6 (*) BYTE DATA
      ('THE ACTUAL ESTIMATED POSITION IS:$$'),

T(0) = HOURS; /* SAVE TIME OF CALL */
T(1) = MINUTES;
T(2) = SECONDS;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(. TITLE$3);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG3);
  CALL SEND$CRLF;
  DESIG = GET$DESIG;
  INDEX = CHECK$DESIG(DESIG);
  IF INDEX = 0FFH
  THEN DO;
    CALL CRT$PRINT$STRING(. MSG4);
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
  END;
  ELSE DO;
    OK = 1;
  END;
  CALL CLEAR$LOW$SCREEN;
END;
CALL CRT$PRINT$STRING(. TITLE$3);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG$03);
CALL SEND$CRLF;
CALL DISPLAY$DESIG(DESIG);
```

DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```

CALL SEND$SPACE(15);
CALL DISPLAY$TYPE(, CONTACT$INFO(INDEX), TYPE);
CALL SEND$SPACE(6);
CALL DISPLAY$CLASS(, CONTACT$INFO(INDEX), KIND);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(, MSG5);
IF CONTACT$INFO(INDEX).FLAG
  THEN TEMP = 15;
  ELSE TEMP = (CONTACT$INFO(INDEX). POINTER MOD 15) + 1;
CALL BYTE$CHAR(TEMP);
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
CALL CRT$PRINT$STRING(, TITLE$3);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(, MSG$00);
CALL SEND$CRLF;
POINTER = CONTACT$INFO(INDEX). POINTER;
CALL CONV$LAT$LONG(, CONTACT$POSI(POINTER), X, CONTACT$POSI(POINTER), Y,
  , LAT, , LONG);
CALL DISPLAY$LAT$LONG(0, , LAT);
CALL SEND$SPACE(12);
CALL DISPLAY$LAT$LONG(1, , LONG);
CALL SEND$CRLF;
CALL DISPLAY$XY(0, , CONTACT$POSI(POINTER), X);
CALL SEND$SPACE(18);
CALL DISPLAY$XY(1, , CONTACT$POSI(POINTER), Y);
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;

```

DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```
CALL CLEAR$LOW$SCREEN;
TEMP = GET$CPA(INDEX, . STRING);
CALL CRT$PRINT$STRING(. TITLE$3);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG$01);
CALL DISPLAY$TIME(. CONTACT$POSI(POINTER). TIME);
CALL SEND$CRLF;
CALL DISPLAY$CRS$BRG(1, . CONTACT$POSI(POINTER). BRG);
CALL SEND$SPACE(20);
CALL DISPLAY$RANGE(. CONTACT$POSI(POINTER). RNG);
CALL SEND$CRLF;
IF CONTACT$INFO(INDEX). CRS$FLAG
THEN DO;
  CALL DISPLAY$CRS$BRG(0, . CONTACT$POSI(POINTER). CRS);
  CALL SEND$SPACE(20);
  END;
ELSE DO;
  CALL CRT$PRINT$STRING(. MSG0);
  CALL SEND$SPACE(3);
  END;
IF CONTACT$INFO(INDEX). SPD$FLAG
THEN CALL DISPLAY$SPD(. CONTACT$POSI(POINTER). SPD);
ELSE CALL CRT$PRINT$STRING(. MSG1);
CALL SEND$CRLF;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
CALL CRT$PRINT$STRING(. TITLE$3);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(. MSG$02);
CALL SEND$CRLF;
```

DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```
IF TEMP = 0
THEN DO;
  CALL CRT$PRINT$STRING(, MSG2);
  CALL SEND$CRLF;
  END;
ELSE DO;
  IF STRING(14) < 3AH
  THEN DO;
    CALL CRT$PRINT$STRING(, 'TIME:  ');
    DO I = 7 TO 10;
      IF I = 9 THEN CALL CRT$WRITE(COLON);
      CALL CRT$WRITE(STRING(I));
    END;
    CALL SEND$SPACE(9);
    CALL CRT$PRINT$STRING(, 'BEARING:  ');
    DO I = 11 TO 14;
      IF I = 14 THEN CALL CRT$WRITE(POINT);
      CALL CRT$WRITE(STRING(I));
    END;
    CALL SEND$SPACE(9);
    CALL CRT$PRINT$STRING(, 'RANGE:  ');
    DO I = 15 TO 19;
      CALL CRT$WRITE(STRING(I));
    END;
    IF STRING(20) = 'M'
    THEN CALL CRT$PRINT$STRING(, ' MILES$');
    ELSE CALL CRT$PRINT$STRING(, ' YARDS$');
    CALL SEND$CRLF;
    END;
  ELSE DO;
    CALL SEND$SPACE(30);
```


DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```

IF STRING(14) = 'L'
THEN DO;
    CALL SEND$SPACE(5);
    CALL START$BLINK;
    CALL CRT$PRINT$STRING(, 'COLLISION AT $$', 1);
    STRING(12), STRING(13) = '$';
    STRING(11) = STRING(10);
    STRING(10) = STRING(9);
    STRING(9) = COLON;
    CALL CRT$PRINT$STRING(, STRING(7));
    CALL CRT$WRITE(18H);
    END;
ELSE DO;
    IF STRING(14) = 'A'
    THEN CALL CRT$PRINT$STRING(, 'MOVING AWAY, $$', 1);
    ELSE CALL CRT$PRINT$STRING(, 'SAME COURSE AND SPEED, $$', 1);
    END;
END;

END;
CALL SEND$CRLF;
CALL CHECK$GO$KEY;
CALL CLEAR$LOW$SCREEN;
CALL CRT$PRINT$STRING(, TITLE$3);
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(, MSG6);
CALL SEND$CRLF;
IF CONTACT$INFO(, INDEX), CRS$FLAG AND
CONTACT$INFO(, INDEX), SPD$FLAG
THEN DO;
    IF T(0) < CONTACT$POSI(, POINTER), TIME(0)
    THEN T(0) = T(0) + 24;

```

DISPLAY\$CMDS

DISPLAY\$CONTACT\$INFO

```

/* FIND ESTIMATED TIME */
CALL CONV$CONTACT$TIME(.T, .TIME$1);
CALL CONV$CONTACT$TIME(.CONTACT$POSI(POINTER).TIME, .TIME$2);
CALL FSUB(.TIME$1, .TIME$2, .TIME$1);
/* FIND ESTIMATED DISTANCE TRAVELED BY CONTACT */
CALL FMUL(.CONTACT$POSI(POINTER).SPD, .TIME$1, .DISTANCE);
CALL FMUL(.DEG$TO$RAD, .CONTACT$POSI(POINTER).CRS, .BRG);
CALL COS$SIN(.BRG, .COS$DIST, .SIN$DIST);
CALL FMUL(.DISTANCE, .COS$DIST, .Y$DELTA);
CALL FMUL(.DISTANCE, .SIN$DIST, .X$DELTA);
CALL FADD(.Y$DELTA, .CONTACT$POSI(POINTER).Y, .Y$DELTA);
CALL FADD(.X$DELTA, .CONTACT$POSI(POINTER).X, .X$DELTA);
TEMP = OWN$SHIP$INFO.POINTER;
CALL FSUB(.Y$DELTA, .OWN$SHIP(TEMP).Y, .Y$DELTA);
CALL FSUB(.X$DELTA, .OWN$SHIP(TEMP).X, .X$DELTA);
/* FIND ESTIMATED BEARING */
CALL ARCTAN(.Y$DELTA, .X$DELTA, .BRG);
CALL FDIV(.BRG, .DEG$TO$RAD, .BRG);
/* FIND ESTIMATED RANGE */
CALL FSQR(.Y$DELTA, .Y$DELTA);
CALL FSQR(.X$DELTA, .X$DELTA);
CALL FADD(.Y$DELTA, .X$DELTA, .RNG);
CALL FSQRT(.RNG, .RNG);
/* PRINT ESTIMATED VALUES */
CALL DISPLAY$CRS$BRG(1, .BRG);
CALL SEND$SPACE(20);
CALL DISPLAY$RANGE(.RNG);
END;
ELSE DO;
CALL CRT$PRINT$STRING(.MSG0);
CALL SEND$SPACE(3);

```

DISPLAY\$CONTACT\$INFO

DISPLAY\$CMDS

```
CALL CRT$PRINT$STRING(, MSG1);  
END;  
CALL SEND$CRLF;  
CALL SEND$CRLF;  
CALL CHECK$GO$KEY;  
CALL CLEAR$LOW$SCREEN;  
END DISPLAY$CONTACT$INFO;
```

DISPLAY\$CMDS

DISPLAY\$SYSTEM

```

/*****
*
* DISPLAY$SYSTEM:
* THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATIONS OF ALL THE CONTACTS
* THAT ARE BEING MAINTAINED BY THE SYSTEM.
*
*****/
DISPLAY$SYSTEM: PROCEDURE PUBLIC;
  DCL BUFFER (18) BYTE,
  I BYTE;
  DCL M0 (*) BYTE DATA
  ('THE FOLLOWING CONTACTS ARE BEING MAINTAINED BY THE SYSTEM:$$');

  DO I = 0 TO LAST(BUFFER);
    BUFFER(I) = ' ';
  END;
  BUFFER(16), BUFFER(17) = '$';
  CALL CRT$PRINT$STRING(, TITLE$4);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(, M0);
  CALL SEND$CRLF;
  DO I = 0 TO 14;
    IF CONTACT$INFO(I).DESIG <> 00H
      THEN DO;
        CALL DE$HASH(CONTACT$INFO(I).DESIG, . BUFFER(10));
        CALL CRT$PRINT$STRING(, BUFFER);
      END;
  END;
  CALL SEND$CRLF;
  CALL CHECK$GO$KEY;
  CALL CLEAR$LOW$SCREEN;

```


DISPLAY\$SYSTEM

DISPLAY\$CMDS

END DISPLAY\$SYSTEM;

DISPLAY\$CMDS

DISPLAY\$SAFE\$RNG

```
/******  
* DISPLAY$SAFE$RNG:  
* THIS PROCEDURE IS USED TO PRESENT TO THE OPERATOR THE CURRENT VALUE OF  
* THE SAFE C. P. A. RANGE.  
*  
*****/  
DISPLAY$SAFE$RNG: PROCEDURE PUBLIC;  
CALL CRT$PRINT$STRING(. TITLE$5);  
CALL SEND$CRLF;  
CALL SEND$CRLF;  
CALL DISPLAY$RANGE(. SAFE$RNG);  
CALL SEND$CRLF;  
CALL SEND$CRLF;  
CALL CHECK$GO$KEY;  
CALL CLEAR$LOW$SCREEN;  
END DISPLAY$SAFE$RNG;
```

DISPLAY\$CMDS

DISPLAY\$WIND

```

/*****
*
* DISPLAY$WIND:
* THIS PROCEDURE IS USED TO DISPLAY INFORMATION ABOUT THE WIND.
*
*****/
*****
DISPLAY$WIND: PROCEDURE PUBLIC;
    DCL STRING (6) BYTE,
        <TEMP, I> BYTE;
    DCL MSG0 (*) BYTE DATA <'WIND DIRECTION: '$'\>,
        MSG1 (*) BYTE DATA <'WIND SPEED: '$'\>;

    CALL CRT$PRINT$STRING(<. TITLE$6>);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(<. MSG0>);
    TEMP = FP$FORMAT(<. SYSTEM.WIND$DIR, . STRING, 3, 1>);
    DO I = 0 TO 3;
        IF I = 3 THEN CALL CRT$WRITE(<POINT>);
        CALL CRT$WRITE(<STRING(I)>);
    END;
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(<. MSG1>);
    TEMP = FP$FORMAT(<. SYSTEM.WIND$SPD, . STRING, 2, 1>);
    DO I = 0 TO 2;
        IF I = 2 THEN CALL CRT$WRITE(<POINT>);
        CALL CRT$WRITE(<STRING(I)>);
    END;
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;

```

DISPLAY\$WIND

DISPLAY\$CMDS

CALL CLEAR\$LOW\$SCREEN;
END DISPLAY\$WIND;

DISPLAY\$CMDS

DISPLAY\$UPDATE\$TIME

```

/*****
*
* DISPLAY$UPDATE$TIME:
* THIS PROCEDURE IS USED TO DISPLAY THE TIME BETWEEN UPDATES.
*
*****/
DISPLAY$UPDATE$TIME: PROCEDURE (TEMP$TIME) PUBLIC;
    DCL TEMP$TIME BYTE;
    DCL M0 (*) BYTE DATA
        ('THE TIME BETWEEN UPDATES IS $$'),
    M1 (*) BYTE DATA
        (' SECONDS. $$');

    CALL CRT$PRINT$STRING(, TITLE$7);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(, M0);
    CALL BYTE$CHAR(TEMP$TIME);
    CALL CRT$PRINT$STRING(, M1);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    CALL CLEAR$LOW$SCREEN;
    END DISPLAY$UPDATE$TIME;

END DISPLAY$CMDS;
```

COMMANDS

COMMANDS

COMMANDS: DO;

CRT\$READ:
PROCEDURE BYTE EXTERNAL;
END;

CRT\$PRINT\$STRING:
PROCEDURE (A) EXTERNAL;
DECLARE A ADDRESS; END;

CRT\$WRITE:
PROCEDURE (A) EXTERNAL;
DECLARE A BYTE; END;

SEND\$CR:
PROCEDURE EXTERNAL;
END;

SEND\$BEL:
PROCEDURE EXTERNAL;
END;

SEND\$BS:
PROCEDURE EXTERNAL;
END;

SEND\$SUB:
PROCEDURE EXTERNAL;

COMMANDS

COMMANDS

```

END;

SEND$CRLF:
  PROCEDURE EXTERNAL;
END;

GET$STRING:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE A ADDRESS, B BYTE; END;

PUT$NUMBER$BUFFER:
  PROCEDURE (A,B) EXTERNAL;
  DECLARE A BYTE, B ADDRESS; END;

ASCII$TO$FLOAT:
  PROCEDURE (A,B,C) EXTERNAL;
  DECLARE (A,C) ADDRESS, B BYTE; END;

FLOAT$TO$ASCII:
  PROCEDURE (A,B,C) EXTERNAL;
  DECLARE (A,B,C) ADDRESS; END;

CLEAR$LOW$SCREEN:
  PROCEDURE EXTERNAL;
END;

FADD:
  PROCEDURE (A,B,C) EXTERNAL;
  DECLARE (A,B,C) ADDRESS; END;

FSUB:

```

COMMANDS

COMMANDS

```
PROCEDURE (A,B,C) EXTERNAL;  
DECLARE (A,B,C) ADDRESS; END;
```

```
FMUL:  
PROCEDURE (A,B,C) EXTERNAL;  
DECLARE (A,B,C) ADDRESS; END;
```

```
FDIV:  
PROCEDURE (A,B,C) EXTERNAL;  
DECLARE (A,B,C) ADDRESS; END;
```

```
EDIV:  
PROCEDURE (A,B,C,D) EXTERNAL;  
DECLARE (A,B,C,D) ADDRESS; END;
```

```
FLTDS:  
PROCEDURE (A,B) EXTERNAL;  
DECLARE (A,B) ADDRESS; END;
```

```
FIXSD:  
PROCEDURE (A,B) EXTERNAL;  
DECLARE (A,B) ADDRESS; END;
```

```
FCMPR:  
PROCEDURE (A,B,C) BYTE EXTERNAL;  
DECLARE (A,B) ADDRESS, C BYTE; END;
```

```
FZTST:  
PROCEDURE (A,B) BYTE EXTERNAL;  
DECLARE (A,B) ADDRESS; END;
```


COMMANDS

COMMANDS

DECLARE LIT LITERALLY 'LITERALLY',
DCL LIT 'DECLARE';

DCL POINT LIT '2EH',
ETEOL LIT '17H';

/* DECIMAL POINT. */
/* ERASE TO END OF LINE. */

DCL

FP\$0\$25 (4) BYTE DATA (00H,00H,80H,3EH),
FP\$2 (4) BYTE DATA (00H,00H,00H,40H),
FP\$5 (4) BYTE DATA (00H,00H,0A0H,040H),
FP\$25 (4) BYTE DATA (00H,00H,0C8H,41H),
FP\$60 (4) BYTE DATA (00H,00H,70H,42H),
FP\$90 (4) BYTE DATA (00H,00H,0B4H,42H),
FP\$100 (4) BYTE DATA (00H,00H,0C8H,042H),
FP\$180 (4) BYTE DATA (00H,00H,34H,43H),
FP\$360 (4) BYTE DATA (00FH,0FFH,0B3H,043H),
FP\$2000 (4) BYTE DATA (0E4H,2BH,0FDH,044H),
FP\$202500 (4) BYTE DATA (04AH,0CAH,045H,048H),
FP\$59\$9 (4) BYTE DATA (9AH,99H,6FH,42H),
FP\$99\$9 (4) BYTE DATA (0CDH,0CCH,0C7H,042H),
FP\$MIN\$T0\$RAD (4) BYTE DATA (98H,82H,98H,39H);

/* 2025.3716 */
/* 202537.15625 */

/* 0.00029089 */

COMMANDS

PRINT\$ERROR\$MSG

```

/*****
* PRINT$ERROR$MSG:
* THIS PROCEDURE IS USED TO PRINT AN ERROR MESSAGE FOR INVALID INPUT.
* IT WILL ALSO RETURN THE CURSOR TO THE BEGINNING OF THE SAME LINE.
*
*****/
PRINT$ERROR$MSG: PROCEDURE;
  DCL MSG(*) BYTE DATA (' *** BAD FORMAT. ****$');
  CALL SEND$BEL;
  CALL CRT$PRINT$STRING( MSG);
  CALL SEND$CR;
  CALL SEND$SUB;
  END PRINT$ERROR$MSG;

```

COMMANDS

CHECK\$YES\$NO

```

/*****
*
* CHECK$YES$NO:
* PROCEDURE USED TO CHECK FOR A VALID YES/NO INPUT FROM THE CRT.
*
* USAGE:
* TYPED PROCEDURE. A VALUE OF 1 WILL BE RETURNED IF THE ANSWER IS 'YES',
* OTHERWISE, THE VALUE RETURNED IS 0.
*
*****
CHECK$YES$NO: PROCEDURE BYTE PUBLIC;
DCL CHAR BYTE;
CHAR = CRT$READ;
DO WHILE (CHAR <> 'Y') AND (CHAR <> 'N') /* UPPER CASE. */
AND (CHAR <> 'y') AND (CHAR <> 'n'); /* LOWER CASE. */
CALL SEND$BEL;
CHAR = CRT$READ;
END;
IF (CHAR = 'Y') OR (CHAR = 'y')
THEN DO;
CALL CRT$PRINT$STRING(, ('YES$$'));
RETURN 1;
END;
ELSE DO;
CALL CRT$PRINT$STRING(, ('NO $$'));
RETURN 0;
END;
END CHECK$YES$NO;

```

COMMANDS

CHECK\$FP\$VALUE

```

/*****
*
* CHECK$FP$VALUE:
* THIS PROCEDURE IS USED TO CHECK A GIVEN VALUE AGAINST A GIVEN LIMIT.
*
* PARAMETERS:
* - A - POINTER TO A FP VALUE.
* - B - POINTER TO A FP VALUE DENOTING THE LIMIT.
*
* USAGE:
* TYPED PROCEDURE. A VALUE OF 00H WILL BE RETURNED IF THE VALUE IS GREATER
* THAN THE GIVEN LIMIT. OTHERWISE A VALUE OF 001H WILL BE RETURNED.
*
*****/
CHECK$FP$VALUE: PROCEDURE (A,B) BYTE PUBLIC;
DCL (A,B) ADDRESS,
    (CHECK, TWO) BYTE;
TWO = 2;
IF (CHECK := FCMPR(A,B,.TWO))
THEN DO;
    CALL PRINT$ERROR$MSG;
    RETURN 0;
END;
ELSE DO;
    CALL CRT$WRITE(ETEOL);
    RETURN 1;
END;
END CHECK$FP$VALUE;
/* ERASE TO THE END OF LINE. */

```


CHECK\$INPUT

COMMANDS

```

/*****
*
* CHECK$INPUT:
* PROCEDURE USED TO CHECK THE VALIDITY OF THE INPUT PRESENT AT THAT MOMENT
* IN THE SCREEN.
*
*****/
CHECK$INPUT: PROCEDURE BYTE PUBLIC;
    DCL CHAR BYTE;
    CALL CRT$PRINT$STRING( 'IS THE INPUT CORRECT? (Y/N)  $$$');
    CHAR = CHECK$YES$NO;
    RETURN CHAR;
END CHECK$INPUT;

```

COMMANDS

GET\$DEGREES

```

/*****
*
* GET$DEGREES:
* THIS PROCEDURE IS USED TO OBTAIN 2 OR 3 NUMERIC CHARACTERS FROM THE CRT
* THAT REPRESENT A VALUE IN DEGREES OF LATITUDE OR LONGITUDE RESPECTIVELY.
*
* PARAMETERS:
* - NUM - NUMBER OF NUMERIC CHARACTERS DESIRED. (2 OR 3 ONLY)
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE RESULT IS DESIRED.
*
* USAGE:
* THE FLOATING POINT REPRESENTATION OF THE DEGREES, WILL BE RETURNED CON-
* VERTED TO MINUTES OF LAT OR LONG.
*
*****/
GET$DEGREES: PROCEDURE (NUM,A) PUBLIC;
DCL A ADDRESS,
      NUMBER BASED A (4) BYTE,
      (NUM, OK) BYTE,
      BUFFER(6) BYTE;
BUFFER(0), BUFFER(1) = NUM;
BUFFER(NUM + 2) = 0;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(,('DEGREES: '$$'));
  CALL PUT$NUMBER$BUFFER(NUM, BUFFER(2));
  CALL ASCII$TO$FLOAT(, BUFFER, NUM + 3, NUMBER);
  IF NUM = 2
    THEN OK = CHECK$FP$VALUE(, NUMBER, .FP$90);
    ELSE OK = CHECK$FP$VALUE(, NUMBER, .FP$180);
  END;

```

COMMANDS

```
CALL FMUL(.NUMBER,.FP$60,.NUMBER);  
END GET$DEGREES;
```

GET\$DEGREES

COMMANDS

GET\$MINUTES

```

/*****
*
* GET$MINUTES:
* THIS PROCEDURE IS USED TO GET FROM THE CRT, THREE NUMERIC CHARACTERS RE-
* PRESENTING THE VALUE OF MINUTES. THIS PROCEDURE WILL PROMPT FOR TWO
* INTEGERS AND ONE DECIMAL VALUE.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FP REPRESENTATION OF THE
* CHARACTERS OBTAINED, IS DESIRED TO BE PLACED.
*
*****/
GET$MINUTES: PROCEDURE (A) PUBLIC;
DCL A ADDRESS,
    NUMBER BASED A (4) BYTE,
    BUFFER(6) BYTE,
    OK BYTE;
    BUFFER(0) = 3;
    BUFFER(1) = 2;
    BUFFER(5) = 0;
    OK = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(, ('MINUTES: $$$$'));
        CALL PUT$NUMBER$BUFFER(2, BUFFER(2));
        CALL CRT$WRITE(POINT);
        CALL PUT$NUMBER$BUFFER(1, BUFFER(4));
        CALL ASCII$TO$FLOAT(, BUFFER, 6, , NUMBER);
        OK = CHECK$FP$VALUE(, NUMBER, , FP$59$9);
    END;
END GET$MINUTES;

```


COMMANDS

GET\$SIGN

```

/*****
*
* GET$SIGN:
* THIS PROCEDURE IS USED TO GET FROM THE CRT, A CHARACTER THAT WILL REPRESENT THE N/S LATITUDE OR E/W LONGITUDE.
*
* PARAMETERS:
* - T1,T2 - ASCII CHARACTERS REPRESENTING THE VALUES AGAINST WHICH THE INPUT FROM THE CRT IS DESIRED TO BE COMPARED.
*
* USAGE:
* TYPED PROCEDURE. A VALUE OF 1 WILL BE RETURNED IF THE INPUT FROM THE CRT IS EQUAL TO THE VALUE OF T1, OTHERWISE A VALUE OF 0 WILL BE RETURNED.
*
*****/
GET$SIGN: PROCEDURE (T1,T2) BYTE PUBLIC;
DECL (T1,T2,SIGN) BYTE;
SIGN = 0;
DO WHILE (SIGN <> T1) AND (SIGN <> T2);
CALL CRT$PRINT$STRING(,('SIGN: '$'));
CALL GET$STRING(,SIGN,1);
IF (SIGN <> T1) AND (SIGN <> T2)
THEN CALL PRINT$ERROR$MSG;
ELSE CALL CRT$WRITE(,TEOL);
END;
IF SIGN = T1 THEN RETURN 1;
ELSE RETURN 0;
END GET$SIGN;

```

COMMANDS

FP\$FORMAT

```

/*****
*
* FP$FORMAT:
* THIS PROCEDURE IS USED TO OBTAIN AN SPECIFIED NUMBER OF ASCII CHARAC-
* TERS REPRESENTING A NUMERIC VALUE IN FP FORMAT.
*
* PARAMETERS:
* - A - POINTER TO A 4 BYTE VECTOR CONTAINING A FP NUMBER.
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF ASCII CHARAC-
* TERS IS DESIRED TO BE PLACED.
* - NUMINT - NUMBER OF CHARACTERS REPRESENTING THE DESIRED INTEGER
* PORTION.
* - NUMDEC - SIMILAR TO NUMINT, BUT REPRESENTING THE DECIMAL PORTION
* DESIRED.
*
* USAGE:
* TYPED PROCEDURE. IF THE SIGN OF THE GIVEN FP NUMBER IS POSITIVE, A
* VALUE OF 0 IS RETURNED, OTHERWISE, A 1 IS RETURNED.
*
*****
FP$FORMAT: PROCEDURE (A,B,NUMINT,NUMDEC) BYTE PUBLIC;
    DCL (A,B) ADDRESS,
        FP$NUM BASED A (4) BYTE,
        STRING BASED B (16) BYTE,
        BUFFER (28) BYTE,
        (NUMINT,NUMDEC,NUM,SIGN,I,J,N,N1) BYTE;
    N1 = 0;
    DO I = 0 TO NUMINT + NUMDEC;
        STRING (I) = '0';
    END;
    CALL FLOAT$TO$ASCII(,FP$NUM,BUFFER,NUM);

```

COMMANDS

```

DO I = NUM TO 27;
  BUFFER(I) = '0';
END;
IF BUFFER(0) = ' '
  THEN SIGN = 0;
  ELSE SIGN = 1;
I = 1;
DO WHILE BUFFER(I) <> POINT;
  N1 = N1 + 1;
  I = I + 1;
END;
IF N1 > NUMINT
  THEN DO;
    CALL CRT$PRINT$STRING( ('NUMBER TOO LARGE. $$$'));
    HALT;
    END;
I = 1;
J = NUMINT - N1;
N = NUMINT + NUMDEC;
DO WHILE N > 0;
  IF BUFFER(I) = POINT THEN I = I + 1;
  STRING(J) = BUFFER(I);
  J = J + 1;
  I = I + 1;
  N = N - 1;
END;
IF (BUFFER(I) >= '5') AND (STRING(J-1) <> '9') /* TO "ROUND". */
  THEN STRING(J-1) = STRING(J-1) + 1;
N = NUMINT + NUMDEC;
STRING(N), STRING (N + 1) = '$';
RETURN SIGN;

```

FP\$FORMAT

COMMANDS

END FP\$FORMAT;

COMMANDS

RANGE\$FORMAT

```

/*****
*
* RANGE$FORMAT:
* THIS PROCEDURE IS USED TO OBTAIN IN ASCII CHARACTERS, THE VALUE OF A
* FP NUMBER REPRESENTING A RANGE. IF THE VALUE IS LESS THAN OR EQUAL TO 5
* MILES, THEN THE RANGE WILL BE GIVEN IN YARDS, OTHERWISE, IT WILL BE GIVEN
* IN MILES.
*
* PARAMETERS:
* - A - POINTER TO A 4 BYTE VECTOR REPRESENTING A FP NUMBER.
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO BE
* PLACED.
*
*****/
RANGE$FORMAT: PROCEDURE (A,B) PUBLIC;
  DCL (A,B) ADDRESS,
    VALUE BASED A (4) BYTE,
    STRING BASED B (6) BYTE,
    RESULT (4) BYTE,
    (TEMP, TWO) BYTE;
  TWO = 2;
  IF (TEMP := FCNPR(A, FP$5, TWO))
  THEN DO;
    TEMP = FP$FORMAT(.VALUE, .STRING(0), 3, 1);
    STRING(4) = STRING(3);
    STRING(3) = POINT;
    STRING(5) = 'M';
  END;
  ELSE DO;
    CALL FMUL(.VALUE, FP$2000, RESULT);
    TEMP = FP$FORMAT(.RESULT, .STRING(0), 5, 0);
  
```

COMMANDS

```
STRING(5) = 'Y';  
END;  
END RANGE$FORMAT;
```

RANGE\$FORMAT

COMMANDS

LAT\$LONG\$FORMAT

```

/*****
*
* LAT$LONG$FORMAT:
* THIS PROCEDURE IS USED TO CONVERT A FLOATING POINT VALUE REPRESENTATION
* OF LAT/LONG IN MINUTES, INTO A CORRESPONDING STRING OF CHARACTERS.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION CONTAINING A FP VALUE.
* - B - POINTER TO A MEMORY LOCATION IN WHICH THE STRING OF ASCII CHARAC-
* TERS IS DESIRED TO BE PLACED.
* - TYPE - CAN HAVE ONE OF TWO VALUES: 0 ---> LATITUDE DESIRED.
* 1 --> LONGITUDE DESIRED.
*
*****/
LAT$LONG$FORMAT: PROCEDURE (A,B,TYPE) PUBLIC;
DCL (A,B) ADDRESS,
SIXTY ADDRESS DATA (060),
LAT$LONG BASED A (4) BYTE,
STRING BASED B (9) BYTE,
BUFFER(28) BYTE,
VALUE (4) BYTE,
TEMP (4) BYTE,
TEMP1 (4) BYTE,
REM (4) BYTE,
(NUM,I,SIGN,TYPE,J,TEST) BYTE;
SIGN = 0;
DO I = 0 TO 3;
VALUE(I) = LAT$LONG(I);
END;
IF LAT$LONG(3) >= 080H
THEN DO;

```

COMMANDS

LAT\$LONG\$FORMAT

```

SIGN = 1;
VALUE(3) = VALUE(3) XOR 080H;
END;

CALL FIXDC( VALUE, . TEMP);
CALL EDIV( TEMP, . SIXTY, . TEMP1, . REM);
CALL FLTDC( TEMP1, . TEMP1);
TEST = FP$FORMAT( TEMP1, . STRING, 3, 0);
CALL FLTDC( REM, . REM);
CALL FLTDC( TEMP, . TEMP);
CALL FSUB( VALUE, . TEMP, . TEMP);
CALL FADD( TEMP, . REM, . TEMP);
TEST = FP$FORMAT( TEMP, . STRING(3), 2, 1);
IF TYPE = 0
THEN DO;
  IF SIGN = 1 THEN STRING(6) = 'S';
  ELSE STRING(6) = 'N';
END;
ELSE DO;
  IF SIGN = 1 THEN STRING(6) = 'W';
  ELSE STRING(6) = 'E';
END;
STRING(7), STRING(8) = '$';
END LAT$LONG$FORMAT;

```


COMMANDS

GET\$TIME\$ZONE

```

/*****
*
* GET$TIME$ZONE:
* THIS PROCEDURE IS USED TO OBTAIN A STRING OF ASCII CHARACTERS FROM
* THE CRT, REPRESENTING THE VALUE AND SIGN OF THE TIME ZONE.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO
* BE PLACED.
*
*****/
GET$TIME$ZONE: PROCEDURE (A) PUBLIC;
    DCL A ADDRESS,
        RESULT BASED A (5) BYTE,
        (OK, SIGN, TEMP, VALUE) BYTE;
    OK = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(, ('ENTER THE TIME ZONE VALUE AS REQUESTED:$$'));
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(, ('SIGN: $$'));
        SIGN = CRT$READ;
        DO WHILE (SIGN <> '+' ) AND (SIGN <> '-');
            CALL SEND$BEL;
            SIGN = CRT$READ;
        END;
        RESULT(0) = SIGN;
        CALL CRT$WRITE(SIGN);
        CALL SEND$CRLF;
        TEMP = 0;
        DO WHILE TEMP = 0;
            CALL CRT$PRINT$STRING(, ('VALUE: $$'));

```

COMMANDS

GET\$TIME\$ZONE

```
CALL PUT$NUMBER$BUFFER(2, RESULT(1));
VALUE = (<RESULT(1) - 30H> * 10) + (<RESULT(2) - 30H>);
IF VALUE > 12 THEN CALL PRINT$ERROR$MSG;
ELSE DO;
  CALL CRT$WRITE(ETEOL);
  TEMP = 1;
  END;
END;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
RESULT(3), RESULT(4) = '$';
END GET$TIME$ZONE;
```

COMMANDS

GET\$LAT

```

/*****
*
* GET$LAT:
*   PROCEDURE USED TO OBTAIN THE VALUES OF LATITUDE.
*
* PARAMETERS:
*   - A - POINTER TO MEMORY LOCATION IN WHICH THE FP REPRESENTATION
*     OF THE VALUE OF THE LATITUDE IN MINUTES IS DESIRED TO BE PLACED.
*
*****/
GET$LAT: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        RESULT BASED A (4) BYTE,
        OP1 (4) BYTE,
        OP2 (4) BYTE,
        <SIGN,OK,TEST,FIVE> BYTE;
  FIVE = 5;
  OK = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(<'ENTER THE LATITUDE VALUE AS REQUESTED:$$'>);
    CALL SEND$CRLF;
    CALL GET$DEGREES(2, OP1);
    CALL SEND$CRLF;
    CALL GET$MINUTES(, OP2);
    CALL SEND$CRLF;
    CALL FADD(, OP1, OP2, RESULT);
    SIGN = GET$SIGN('N', 'S');
    IF SIGN = 0
      THEN DO;
        IF <TEST:= FZTST(RESULT, FIVE)> THEN
          RESULT(3) = RESULT(3) XOR 080H;

```

GET\$LAT

COMMANDS

```
CALL CRT$PRINT$STRING<.<'OUTH. $$'>>
END;
ELSE CALL CRT$PRINT$STRING<.<'ORTH. $$'>>
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
END GET$LAT;
```


COMMANDS

GET\$LONG

```

/*****
*
* GET$LONG:
* PROCEDURE USED TO OBTAIN THE FP REPRESENTATION OF THE LONGITUDE IN
* MINUTES.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE VALUE OF THE
* LONGITUDE IN MINUTES IS DESIRED TO BE STORED.
*
*****/
GET$LONG: PROCEDURE (A) PUBLIC;
DCL A ADDRESS,
      RESULT BASED A (4) BYTE,
      OP1 (4) BYTE,
      OP2 (4) BYTE,
      <SIGN,OK,TEST,FIVE> BYTE;
FIVE = 5;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING<('ENTER THE LONGITUDE VALUE AS REQUESTED:$$')>;
  CALL SEND$CRLF;
  CALL GET$DEGREES<3, OP1>;
  CALL SEND$CRLF;
  CALL GET$MINUTES<OP2>;
  CALL SEND$CRLF;
  CALL FADD<OP1, OP2, RESULT>;
  SIGN = GET$SIGN<'E','W'>;
  IF SIGN = 0
  THEN DO,
    CALL CRT$PRINT$STRING<('EST$$')>;

```

COMMANDS

GET\$LONG

```
IF <TEST:= FZTST<.RESULT,.FIVE>> THEN
  RESULT<3> = RESULT<3> XOR 080H;
END;
ELSE CALL CRT$PRINT$STRING<('AST$$')>;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
END GET$LONG;
```

COMMANDS

GET\$COURSE\$BRG

```

/*****
*
* GET$COURSE$BRG:
* THIS PROCEDURE IS USED TO OBTAIN THE VALUE OF COURSE OR BEARING FROM
* THE CRT.
*
* PARAMETERS:
* - TYPE. - HAS TWO POSSIBLE VALUES: IF 0 ---> GET COURSE.
*           IF 1 ---> GET BEARING.
* - A. - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESENTATION OF THE VALUE OF THE COURSE OR BEARING, IN DEGREES, IS DESIRED TO BE PLACED.
*
*****
GET$COURSE$BRG: PROCEDURE (TYPE,A) PUBLIC;
    DCL A ADDRESS,
        RESULT BASED A (4) BYTE,
        BUFFER(7) BYTE,
        (TEMP, OK, TYPE) BYTE;
    OK = 0;
    BUFFER(0) = 4;
    BUFFER(1) = 3;
    BUFFER(6) = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(,('ENTER THE $$'));
        IF TYPE = 0
            THEN CALL CRT$PRINT$STRING(,('COURSE $$'));
            ELSE CALL CRT$PRINT$STRING(,('BEARING $$'));
        CALL CRT$PRINT$STRING(,('VALUE AS REQUESTED: $$'));
        CALL SEND$CRLF;
        TEMP = 0;

```

COMMANDS

GET\$COURSE\$BRG

```
DO WHILE TEMP = 0;
  CALL CRT$PRINT$STRING(,'DEGREES: '$');
  CALL PUT$NUMBER$BUFFER(3,.BUFFER(2));
  CALL CRT$WRITE(POINT);
  CALL PUT$NUMBER$BUFFER(1,.BUFFER(5));
  CALL ASCII$TO$FLOAT(.BUFFER,7,.RESULT);
  TEMP = CHECK$FP$VALUE(.RESULT,.FP$360);
END;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
END GET$COURSE$BRG;
```


COMMANDS

GET\$SPEED

```

/*****
*
* GET$SPEED:
*   THIS PROCEDURE IS USED TO OBTAIN THE SPEED VALUE FROM THE CRT.
*
* PARAMETERS:
*   - A - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESENTATION OF THE VALUE OF THE SPEED IN KNOTS, IS DESIRED TO BE PLACED.
*
*****/
GET$SPEED: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        RESULT BASED A (4) BYTE,
        BUFFER(6) BYTE,
        (TEMP,OK) BYTE;

  OK = 0;
  BUFFER(0) = 3;
  BUFFER(1) = 2;
  BUFFER(5) = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(, ('ENTER THE SPEED VALUE AS REQUESTED:$$'));
    CALL SEND$CRLF;
    TEMP = 0;
    DO WHILE TEMP = 0;
      CALL CRT$PRINT$STRING(, ('KNOTS: $$'));
      CALL PUT$NUMBER$BUFFER(2, BUFFER(2));
      CALL CRT$WRITE(POINT);
      CALL PUT$NUMBER$BUFFER(1, BUFFER(4));
      CALL ASCII$TO$FLOAT(, BUFFER, 6, , RESULT);
      TEMP = CHECK$FP$VALUE(, RESULT, , FP$9$9);
    END;
  END;

```

GET\$SPEED

COMMANDS

```
CALL SEND$CRLF;  
OK = CHECK$INPUT;  
CALL CLEAR$LOW$SCREEN;  
END;  
END GET$SPEED;
```

COMMANDS

GET\$RANGE

```

/*****
*
* GET$RANGE:
* THIS PROCEDURE IS USED TO OBTAIN THE FLOATING POINT REPRESENTATION OF
* A RANGE VALUE OBTAINED FROM THE CRT.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESENTATION OF THE VALUE OF A RANGE, IS DESIRED TO BE PLACED.
*
* USAGE:
* ALTHOUGH THIS PROCEDURE WILL ACCEPT EITHER YARDS OR MILES AS UNITS OF RANGE, THE RESULT WILL BE GIVEN IN TERMS OF MILES ONLY.
*
*****/
GET$RANGE: PROCEDURE (A) PUBLIC;
    DCL A ADDRESS,
        RESULT BASED A (4) BYTE,
        BUFFER (8) BYTE,
        <TEMP, OK, CHAR, UNITS> BYTE;
    OK = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(, <'ENTER THE RANGE VALUE AS REQUESTED:$$/>);
        CALL SEND$CRLF;
        CALL CRT$PRINT$STRING(, <'ENTER THE UNITS TO BE USED: (M/Y) $$/>);
        CHAR = CRT$READ;
        DO WHILE (CHAR <> 'Y') AND (CHAR <> 'M') /* UPPER CASE CHARACTERS */
            AND (CHAR <> 'y') AND (CHAR <> 'm'); /* LOWER CASE CHARACTERS */
        CALL SEND$BEL;
        CHAR = CRT$READ;
    END;

```

COMMANDS

GET\$RANGE

```

IF CHAR >= 61H THEN CHAR = CHAR - 020H;
IF CHAR = 'Y' THEN CALL CRT$PRINT$STRING(, ('YARDS, $$'));
ELSE CALL CRT$PRINT$STRING(, ('MILES, $$'));
CALL SEND$CRLF;
TEMP = 0;
DO WHILE TEMP = 0;
IF CHAR = 'Y'
THEN DO;
CALL CRT$PRINT$STRING(, ('YARDS: $$'));
CALL PUT$NUMBER$BUFFER(6, ., BUFFER(2));
BUFFER(0), BUFFER(1) = 6;
BUFFER(8) = 0;
CALL ASCII$TO$FLOAT(, BUFFER, 9, ., RESULT);
TEMP = CHECK$FP$VALUE(, RESULT, ., FP$202500);
CALL FDI$V(, RESULT, ., FP$2000, ., RESULT);
END;
ELSE DO;
CALL CRT$PRINT$STRING(, ('MILES: $$'));
CALL PUT$NUMBER$BUFFER(3, ., BUFFER(2));
CALL CRT$WRITE(POINT);
CALL PUT$NUMBER$BUFFER(1, ., BUFFER(5));
BUFFER(0) = 4;
BUFFER(1) = 3;
BUFFER(6) = 0;
CALL ASCII$TO$FLOAT(, BUFFER, 7, ., RESULT);
TEMP = CHECK$FP$VALUE(, RESULT, ., FP$100);
END;
END;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;

```


COMMANDS

END;
END GET\$RANGE;

GET\$RANGE

COMMANDS

GET\$DESIG

```

/*****
*
* GET$DESIG:
* THIS PROCEDURE IS USED TO OBTAIN THE DESIGNATION OF THE CONTACT FROM
* THE CRT.
*
* USAGE:
* TYPED PROCEDURE. RETURNS A HASHED VALUE TO BE USED INTERNALLY ACCORDING
* TO THE FOLLOWING RULE:
*      HASH = CHAR*100 + CHAR1.
*
*****/
GET$DESIG: PROCEDURE ADDRESS PUBLIC;
DCL HASH ADDRESS,
    BUFFER (2) BYTE,
    (CHAR, OK, CHAR1) BYTE;
OK = 0;
DO WHILE OK = 0;
    CALL CRT$PRINT$STRING( ('DESIG:  $$$'));
    L: CHAR = CRT$READ;
    DO WHILE ((CHAR < 41H) AND (CHAR <> 20H)) OR
        ((CHAR > 5AH) AND (CHAR < 61H)) OR (CHAR > 7AH);
        CALL SEND$BEL;
        CHAR = CRT$READ;
    END;
    IF CHAR >= 61H THEN CHAR = CHAR - 20H;
    CALL CRT$WRITE(CHAR);
    CHAR1 = CRT$READ;
    DO WHILE ((CHAR1 < 41H) OR ((CHAR1 > 5AH) AND
        (CHAR1 < 61H)) OR (CHAR1 > 7AH)) AND (CHAR1 <> 07FH);
        CALL SEND$BEL;

```

COMMANDS

GET\$DESIG

```
CHAR1 = CRT$READ;  
END;  
IF CHAR1 = 07FH THEN DO;  
  CALL SEND$BS;  
  GO TO L;  
END;  
IF CHAR1 >= 61H THEN CHAR1 = CHAR1 - 20H;  
CALL CRT$WRITE(CHAR1);  
CALL SEND$CRLF;  
OK = CHECK$INPUT;  
CALL CLEAR$LOW$SCREEN;  
END;  
HASH = CHAR*100 + CHAR1;  
RETURN HASH;  
END GET$DESIG;
```

COMMANDS

GET\$TYPE

```

/*****
*
* GET$TYPE:
* THIS PROCEDURE IS USED TO OBTAIN THE TYPE OF THE CONTACT FROM
* THE CRT.
*
* USAGE:
* TYPED PROCEDURE. RETURNS THE FOLLOWING VALUES :
* 0 ---> SURFACE.
* 1 ---> SUB-SURFACE.
*
*****/
GET$TYPE: PROCEDURE BYTE PUBLIC;
DCL (TYPE, CHAR, OK) BYTE;
OK = 0;
DO WHILE OK = 0;
CALL CRT$PRINT$STRING(, ('IS IT A SURFACE TYPE CONTACT? (Y/N) $$'));
CHAR = CHECK$YES$NO;
CALL SEND$CRLF;
CALL CRT$PRINT$STRING(, ('TYPE: $$'));
IF CHAR = 1
THEN DO;
CALL CRT$PRINT$STRING(, ('SURFACE. $$'));
TYPE = 0;
END;
ELSE DO;
CALL CRT$PRINT$STRING(, ('SUB-SURFACE. $$'));
TYPE = 1;
END;
CALL SEND$CRLF;
OK = CHECK$INPUT;

```


GET\$TYPE

COMMI INDS

```
CALL CLEAR$LOW$SCREEN;  
END;  
RETURN TYPE;  
END GET$TYPE;
```

COMMANDS

GET\$KIND

```

/*****
*
* GET$KIND:
* THIS PROCEDURE IS USED TO OBTAIN THE CONTACT CLASS FROM THE CRT.
*
* USAGE:
* TYPED PROCEDURE. RETURNS THE FOLLOWING VALUES:
* 0 ---> FRIEND.
* 1 ---> HOSTILE.
* 2 ---> UNKNOWN.
*
*****/
GET$KIND: PROCEDURE BYTE PUBLIC;
DCL (KIND, TEMP, CHAR, OK) BYTE;
OK = 0;
DO WHILE OK = 0;
TEMP = 0;
DO WHILE TEMP = 0;
CALL CRT$PRINT$STRING(, ('ENTER THE CONTACT CLASS: (F/H/U) $$'));
CALL GET$STRING(, CHAR, 1);
IF (CHAR <> 'F') AND (CHAR <> 'H') AND (CHAR <> 'U')
THEN CALL PRINT$ERROR$MSG;
ELSE DO;
CALL CRT$WRITE(, (TEMP));
TEMP = 1;
IF CHAR = 'F'
THEN DO;
CALL CRT$PRINT$STRING(, ('FRIEND. $$'));
KIND = 0;
END;
ELSE DO;

```

COMMANDS

GET\$KIND

```

IF CHAR = 'H'
THEN DO;
  CALL CRT$PRINT$STRING( ('OSTILE. $$'));
  KIND = 1;
END;
ELSE DO;
  CALL CRT$PRINT$STRING( ('UNKNOWN. $$'));
  KIND = 2;
END;

END;

END;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
RETURN KIND;
END GET$KIND;

```

COMMANDS

GET\$SCALE

```

/*****
*
* GET$SCALE:
* THIS PROCEDURE IS USED TO OBTAIN A NUMBER REPRESENTING THE SCALE DESIRED
* TO BE USED IN THE PLOTTING AT THE PLASMA DISPLAY.
*
* PARAMETERS:
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE FLOATING POINT REPRESENTATION OF THE SCALE (MILES/INCH) IS DESIRED TO BE PLACED.
*
*****/
GET$SCALE: PROCEDURE (A) PUBLIC;
    DCL A ADDRESS,
        SCALE BASED A (4) BYTE,
        BUFFER (7) BYTE,
        (TEMP, TEMP1, OK) BYTE;

    OK = 0;
    DO WHILE OK = 0;
        CALL CRT$PRINT$STRING(, ('ENTER THE SCALE AS REQUESTED: $$$'));
        CALL SEND$CRLF;
        TEMP, TEMP1 = 0;
        DO WHILE (TEMP = 0) OR (TEMP1 = 0);
            CALL CRT$PRINT$STRING(, ('MILES PER INCH: $$$'));
            CALL PUT$NUMBER$BUFFER(2, , BUFFER(2));
            CALL CRT$WRITE(POINT);
            CALL PUT$NUMBER$BUFFER(2, , BUFFER(4));
            BUFFER(0) = 4;
            BUFFER(1) = 2;
            BUFFER(6) = 0;
            CALL ASCII$TO$FLOAT(, BUFFER, 7, , SCALE);
            TEMP = CHECK$FP$VALUE(, SCALE, , FP$25);
        END DO;
    END DO;
END GET$SCALE;

```


COMMANDS

GET\$SCALE

```
IF TEMP <> 0 THEN
  TEMP1 = CHECK$FP$VALUE(. FP$0$25, . SCALE);
END;
CALL SEND$CRLF;
OK = CHECK$INPUT;
CALL CLEAR$LOW$SCREEN;
END;
END GET$SCALE;

END COMMANDS;
```

PLASMA\$MODULE

PLASMA\$MODULE

PLASMA\$MODULE: DO;

\$NOLIST

\$INCLUDE (:F1:EXTER. SRC)

\$INCLUDE (:F1:EXTER1. SRC)

\$INCLUDE (:F1:EXTER2. SRC)

\$LIST

PLASMA\$MODULE

DECLARATIONS

```

/***  DECLARATIONS:  ***/

DCL TRUE LIT '0FFH',
FALSE LIT '00H',
EOL LIT '024H',
POINT LIT '02EH';

DCL FP$2  (4) BYTE DATA (000H, 000H, 000H, 040H), /* 2.0 */
FP$8  (4) BYTE DATA (0CDH, 0CCH, 008H, 041H), /* 8.55 */
FP$511 (4) BYTE DATA (000H, 080H, 0FFH, 043H); /* 511.0 */

DCL LAST$POSI (15) STRUCTURE(
    FLAG BYTE,
    X ADDRESS,
    Y ADDRESS);

OS$LAST$POSI STRUCTURE(
    FLAG BYTE,
    X ADDRESS,
    Y ADDRESS);

DCL WINDOW (4) BYTE,
HALF$WINDOW (4) BYTE,
ORIGIN$X (4) BYTE,
ORIGIN$Y (4) BYTE;

```

PLASMA\$MODULE

DECLARATIONS

DCL BUFFER (*) BYTE INITIAL ('SCALE: '))

PLASMA\$MODULE

SET\$WINDOW

```
/******  
* SET$WINDOW:  
* THIS PROCEDURE IS USED TO FIND THE VALUES OF THE WINDOW AND HALF$WINDOW  
* FLOATING POINT VECTORS.  
*  
*****  
SET$WINDOW: PROCEDURE PUBLIC;  
  CALL FMUL(.SYSTEM.SCALE, .FP$8, .WINDOW);  
  CALL FDIV(.WINDOW, .FP$2, .HALF$WINDOW);  
  END SET$WINDOW;
```

PLASMA\$MODULE

CLEAR\$STRUCTURES

```

/*****
*
* CLEAR$STRUCTURES:
* THIS PROCEDURE IS USED TO CLEAR THE STRUCTURES USED IN THIS MODULE.
*
*****/
CLEAR$STRUCTURES:: PROCEDURE PUBLIC;
  DCL A ADDRESS,
        VALUE BASED A BYTE,
        I BYTE;
  A = .LAST$POS1;
  DO I = 0 TO 74;
    VALUE = 00H;
    A = A + 1;
  END;
  A = .OS$LAST$POS1;
  DO I = 0 TO 4;
    VALUE = 00H;
    A = A + 1;
  END;
END CLEAR$STRUCTURES;

```

PLASMA\$MODULE

DRAW\$FRIENDLY\$SYMBOL

```

/*****
*
* DRAW$FRIENDLY$SYMBOL:
* THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN A GIVEN POSITION,
* THE SYMBOL USED TO REPRESENT A FRIENDLY CONTACT.
*
* PARAMETERS:
* - X - ADDRESS VALUE.
* - Y - ADDRESS VALUE.
*
*****
DRAW$FRIENDLY$SYMBOL: PROCEDURE (X, Y) PUBLIC;
  DCL (X, Y) ADDRESS,
    (TEMP$X, TEMP$Y) ADDRESS;
  IF (X < 2) OR (X > 509) OR (Y < 2) OR (Y > 509)
  THEN RETURN; /* SYMBOL CAN NOT BE DRAWN */
  TEMP$X = X - 1;
  TEMP$Y = Y + 2;
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X + 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X + 2;
  TEMP$Y = Y + 1;
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$Y = Y - 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X + 1;
  TEMP$Y = Y - 2;
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X - 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);

```

PLASMA\$MODULE

DRAW\$FRIENDLY\$SYMBOL

```
TEMP$X = X - 2;  
TEMP$Y = Y - 1;  
CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);  
TEMP$Y = Y + 1;  
CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);  
END DRAW$FRIENDLY$SYMBOL
```


PLASMA\$MODULE

DRAW\$UNK\$HOS\$SYMBOL

```

/*****
*
* DRAW$UNK$HOS$SYMBOL:
* THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN A GIVEN POINT,
* THE SYMBOL USED TO REPRESENT UNKNOWN AND/OR HOSTILE CONTACTS.
*
* PARAMETERS:
* - X - ADDRESS VALUE.
* - Y - ADDRESS VALUE.
*
*****/
DRAW$UNK$HOS$SYMBOL: PROCEDURE (X, Y) PUBLIC
  DCL (X, Y, TEMP$X, TEMP$Y) ADDRESS;
  IF (X < 2) OR (X > 509) OR (Y < 2) OR (Y > 509)
    THEN RETURN; /* SYMBOL CAN NOT BE DRAWN. */
  TEMP$X = X - 2;
  TEMP$Y = Y - 2;
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X + 2;
  TEMP$Y = Y + 2;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X - 2;
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X + 2;
  TEMP$Y = Y - 2;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  END DRAW$UNK$HOS$SYMBOL;

```

PLASMA\$MODULE

DRAW\$OWN\$SHIP\$SYMBOL

```

/*****
*
* DRAW$OWN$SHIP$SYMBOL:
* THIS PROCEDURE IS USED TO DRAW IN THE PLASMA DISPLAY, IN A GIVEN POINT,
* THE SYMBOL USED TO REPRESENT THE OWN SHIP.
*
* PARAMETERS:
* - X. - ADDRESS VALUE.
* - Y. - ADDRESS VALUE.
*
*****/
DRAW$OWN$SHIP$SYMBOL: PROCEDURE (X, Y) PUBLIC;
  DCL (X, Y, TEMP$X, TEMP$Y) ADDRESS;
  CALL DRAW$FRIENDLY$SYMBOL(X, Y);
  IF (X = 0) OR (X = 511) OR (Y = 0) OR (Y = 511)
  THEN RETURN; /* SYMBOL CAN NOT BE DRAWN. */
  TEMP$X = X - 1;
  TEMP$Y = Y + 1;
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X + 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$Y = Y - 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$X = X - 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  CALL START$VECTOR$SOLID(TEMP$X, TEMP$Y);
  TEMP$Y = Y + 1;
  CALL STOP$VECTOR$SOLID(TEMP$X, TEMP$Y);
  END DRAW$OWN$SHIP$SYMBOL;

```

PLASMA\$MODULE

CHECK\$PLASMA

```

/*****
*
* CHECK$PLASMA:
* THIS PROCEDURE IS USED TO DETERMINE WHETHER OR NOT A POINT REPRESENTED
* BY TWO X , Y RP VALUES, CAN BE DISPLAYED AT THE PLASMA.
*
* PARAMETERS:
* - A - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE X VALUE;
* - B - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE Y VALUE.
*
* USAGE:
* TYPED PROCEDURE. RETURNS A VALUE OF 'TRUE' (OFFH) IF THE POINT CAN BE
* DISPLAYED AT THE CURRENT SCALE; OTHERWISE, A VALUE OF 'FALSE' (00H)
* IS RETURNED.
*
*****
CHECK$PLASMA: PROCEDURE (A, B) BYTE PUBLIC;
DCL (A, B) ADDRESS,
     X BASED A (4) BYTE,
     Y BASED B (4) BYTE,
     TEMP (4) BYTE,
     CHECK BYTE DATA (02H);
CALL FADD<<.ORIGIN$, .WINDOW, .TEMP);
IF FCMPC<<.X, .TEMP, .CHECK)
THEN RETURN FALSE;
IF FCMPC<<.ORIGIN$, .X, .CHECK)
THEN RETURN FALSE;
CALL FSUB<<.ORIGIN$, .WINDOW, .TEMP);
IF FCMPC<<.Y, .ORIGIN$, .CHECK)
THEN RETURN FALSE;
IF FCMPC<<.TEMP, .Y, .CHECK)

```

PLASMA\$MODULE

CHECK\$PLASMA

```
      THEN RETURN FALSE;  
      /* EVERYTHING IS OKAY. RETURN A TRUE VALUE */  
      RETURN TRUE;  
      END CHECK$PLASMA;
```


PLASMA\$MODULE

NORMALIZE

```

/*****
*
* NORMALIZE:
* THIS PROCEDURE IS USED TO NORMALIZE TWO FP VALUES INTO ADDRESS VALUES
* THAT CAN BE USED TO DETERMINE A POINT IN THE PLASMA DISPLAY. IT MUST
* BE USED AFTER 'CHECK$PLASMA'.
*
* PARAMETERS:
* - A - POINTER TO A FOUR BYTE VECTOR REPRESENTING A FP 'X' VALUE.
* - B - POINTER TO A FOUR BYTE VECTOR REPRESENTING A FP 'Y' VALUE.
* - C - POINTER TO AN ADDRESS VALUE IN WHICH THE 'NORMALIZED X' IS
* DESIRED TO BE PLACED.
* - D - POINTER TO AN ADDRESS VALUE IN WHICH THE 'NORMALIZED Y' IS
* DESIRED TO BE PLACED.
*
*****/
NORMALIZE: PROCEDURE (A, B, C, D) PUBLIC;
  DCL (A, B, C, D) ADDRESS,
  FP$X BASED A (4) BYTE,
  FP$Y BASED B (4) BYTE,
  X BASED C ADDRESS,
  Y BASED D ADDRESS,
  DELTA$X (4) BYTE,
  DELTA$Y (4) BYTE,
  TEMP (4) BYTE,
  TEMP1 (4) BYTE;
  CALL FSUB(.FP$X, .ORIGIN$X, .DELTA$X);
  IF DELTA$X(3) >= 80H
    THEN DELTA$X(3) = DELTA$X(3) XOR 080H; /* ABSOLUTE VALUE REQUIRED */
  CALL FSUB(.ORIGIN$Y, .FP$Y, .DELTA$Y);
  IF DELTA$Y(3) >= 80H

```

PLASMA\$MODULE

NORMALIZE

```

      THEN DELTA$Y(3) = DELTA$Y(3) XOR 080H; /* ABSOLUTE VALUE REQUIRED */
      CALL FDIV(.FP$11, .WINDOW, .TEMP);
      CALL FMUL(.TEMP, .DELTA$X, .TEMP1);
      CALL FIXSD(.TEMP1, .TEMP1);
      X = TEMP1(1);
      X = SHL(X, 8) + TEMP1(0);
      CALL FMUL(.TEMP, .DELTA$Y, .TEMP1);
      CALL FIXSD(.TEMP1, .TEMP1);
      Y = TEMP1(1);
      Y = SHL(Y, 8) + TEMP1(0);
      END NORMALIZE;

```

PLASMA\$MODULE

PUT\$OS\$CENTER

```

/*****
*
* PUT$OS$CENTER:
*   THIS PROCEDURE IS USED TO CHANGE THE X,Y VALUES USED IN THE PLASMA ORI-
*   GIN, IN ORDER TO ALLOW THE OWN SHIP'S LAST POSITION TO BE IN THE NEW
*   PLASMA DISPLAY CENTER.
*
*****/
PUT$OS$CENTER: PROCEDURE PUBLIC;
  DCL POINTER BYTE;
  POINTER = OWN$SHIP$INFO.POINTER;
  CALL FSUBC(OWN$SHIP(POINTER), X, HALF$WINDOW, ORIGIN$X);
  CALL FADD(OWN$SHIP(POINTER), Y, HALF$WINDOW, ORIGIN$Y);
  END PUT$OS$CENTER;

```

PLASMA\$MODULE

PUT\$CONTACT\$CENTER

```

/*****
*
* PUT$CONTACT$CENTER:
*   THIS PROCEDURE IS USED TO CHANGE THE X/Y VALUES USED IN THE PLASMA ORI-
*   GIN, IN ORDER, TO ALLOW THE GIVEN CONTACT LAST POSITION TO BE IN THE NEW
*   PLASMA DISPLAY CENTER.
*
* PARAMETERS:
*   - INDEX - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
*   INFO STRUCTURE.
*
*****/
PUT$CONTACT$CENTER: PROCEDURE (INDEX) PUBLIC
  DCL (POINTER, INDEX) BYTE;
  POINTER = CONTACT$INFO(INDEX).POINTER;
  CALL PSUBC( CONTACT$POS(POINTER).X, HALF$WINDOW, ORIGIN$X);
  CALL PADDC( CONTACT$POS(POINTER).Y, HALF$WINDOW, ORIGIN$Y);
  END PUT$CONTACT$CENTER;

```


PLASMA\$MODULE

FIXED\$REORIENTATION

```

/*****
*
* FIXED$REORIENTATION:
* THIS PROCEDURE IS USED TO DEFINE A NEW REFERENCE POINT FOR THE PLASMA
* DISPLAY, ACCORDING TO 8 PREDEFINED POINTS.
*
* PARAMETERS:
* - TYPE - BYTE NUMBER BETWEEN 0 AND 7 THAT IS USED TO IDENTIFY THE 8 PRE-
*   DEFINED WAYS TO REORIENT THE PLASMA SCREEN.
*
*****/
FIXED$REORIENTATION: PROCEDURE (TYPE) PUBLIC
DCL TYPE BYTE;
DO CASE TYPE;
DO; /* CASE 0 */
CALL FADD(.ORIGIN$Y, .HALF$WINDOW, .ORIGIN$Y);
END;

DO; /* CASE 1 */
CALL FADD(.ORIGIN$, .HALF$WINDOW, .ORIGIN$X);
CALL FADD(.ORIGIN$, .HALF$WINDOW, .ORIGIN$Y);
END;

DO; /* CASE 2 */
CALL FADD(.ORIGIN$, .HALF$WINDOW, .ORIGIN$X);
END;

DO; /* CASE 3 */
CALL FADD(.ORIGIN$, .HALF$WINDOW, .ORIGIN$X);
CALL FADD(.ORIGIN$, .HALF$WINDOW, .ORIGIN$Y);
END;

```

PLASMA\$MODULE

FIXED\$REORIENTATION

```

DO; /* CASE 4 */
  CALL FSUB(. ORIGIN$Y, . HALF$WINDOW, . ORIGIN$Y);
END;

DO; /* CASE 5 */
  CALL FSUB(. ORIGIN$X, . HALF$WINDOW, . ORIGIN$X);
  CALL FSUB(. ORIGIN$Y, . HALF$WINDOW, . ORIGIN$Y);
END;

DO; /* CASE 6 */
  CALL FSUB(. ORIGIN$X, . HALF$WINDOW, . ORIGIN$X);
END;

DO; /* CASE 7 */
  CALL FSUB(. ORIGIN$X, . HALF$WINDOW, . ORIGIN$X);
  CALL FADD(. ORIGIN$Y, . HALF$WINDOW, . ORIGIN$Y);
END;

END; /* END CASE */
END FIXED$REORIENTATION;

```

PLASMA\$MODULE

PLASMA\$REDESIG

```

/*****
*
* PLASMA$REDESIG:
* THIS PROCEDURE IS USED TO DISPLAY, IF POSSIBLE, THE NEW CONTACT'S DESIG
* IN THE LAST KNOWN POSITION.
*
* PARAMETERS:
* - INDEX - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
* INFO STRUCTURE.
*
*****/
PLASMA$REDESIG: PROCEDURE (INDEX) PUBLIC;
    DCL (X, Y) ADDRESS,
        INDEX BYTE;
    IF LAST$POSI<INDEX>.FLAG
    THEN DO,
        X = LAST$POSI<INDEX>.X;
        Y = LAST$POSI<INDEX>.Y;
        CALL GRAPHIC$DESIG(X, Y, CONTACT$INFO<INDEX>.DESIG);
    END;
END PLASMA$REDESIG;

```

PLASMA\$MODULE

PLASMA\$DELETE

```

/*****
*
* PLASMA$DELETE:
* THISPROCEDURE IS USED TO CLEAR THE FLAG CORRESPONDING TO A CONTACT THAT
* HAS BEEN REMOVED FROM THE SYSTEM.
*
* PARAMETERS:
* - INDEX - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
* INFO STRUCTURE.
*
*****/
PLASMA$DELETE: PROCEDURE (INDEX) PUBLIC;
    DCL INDEX BYTE;
    LAST$POS1(INDEX).FLAG = FALSE;
    END PLASMA$DELETE;
/
```


PLASMA\$MODULE

PLASMA\$CONTACT

```

/*****
*
* PLASMA$CONTACT:
* THIS PROCEDURE IS USED TO TRY TO PLOT THE LAST POSITION OF A GIVEN CONTACT-
* ACT, IF POSSIBLE.
*
* PARAMETERS:
* - INDEX - INDICATES THE RELATIVE POSITION OF THE CONTACT IN THE CONTACT$-
* INFO STRUCTURE.
*
*****
PLASMA$CONTACT: PROCEDURE (INDEX) PUBLIC
    DCL (X, Y) ADDRESS,
        (POINTER, INDEX, TEMP) BYTE;
    POINTER = CONTACT$INFO(INDEX).POINTER;
    TEMP = CHECK$PLASMA(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y);
    IF LAST$POSI(INDEX).FLAG
    THEN DO;
        IF TEMP
        THEN DO;
            CALL START$VECTOR$DASH(LAST$POSI(INDEX).X, LAST$POSI(INDEX).Y);
            CALL NORMALIZE(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y,
                .X, .Y);
            CALL STOP$VECTOR$DASH(X, Y);
            IF CONTACT$INFO(INDEX).KIND = 0
            THEN CALL DRAW$FRIENDLY$SYMBOL(X, Y);
            ELSE CALL DRAW$UNK$HOS$SYMBOL(X, Y);
            LAST$POSI(INDEX).X = X;
            LAST$POSI(INDEX).Y = Y;
            END;
        ELSE DO;

```

PLASMA\$MODULE

PLASMA\$CONTACT

```
      LAST$POSI(INDEX).FLAG = FALSE;
    END;

    ELSE DO;
      IF TEMP
      THEN DO;
        CALL NORMALIZE(.CONTACT$POSI(POINTER).X, .CONTACT$POSI(POINTER).Y,
          .X, .Y);
        CALL GRAPHIC$DESIG(X, Y, .CONTACT$INFO(INDEX).DESIG);
        IF .CONTACT$INFO(INDEX).KIND = 0
        THEN CALL DRAW$FRIENDLY$SYMBOL(X, Y);
        ELSE CALL DRAW$UNK$HOS$SYMBOL(X, Y);
        LAST$POSI(INDEX).FLAG = TRUE;
        LAST$POSI(INDEX).X = X;
        LAST$POSI(INDEX).Y = Y;
      END;
    END;

  END;
END PLASMA$CONTACT;
```

PLASMA\$MODULE

PLASMA\$OS

```

/*****
*
* PLASMA$OS:
* THIS PROCEDURE IS USED TO DISPLAY, IF POSSIBLE, THE LAST KNOWN POSITION
* OF THE OWN SHIP IN THE PLASMA DISPLAY.
*
*****
*****
PLASMA$OS: PROCEDURE PUBLIC;
  DCL (X, Y) ADDRESS,
        (POINTER, TEMP) BYTE;
  POINTER = OWN$SHIP$INFO.POINTER;
  TEMP = CHECK$PLASMA(OWN$SHIP(POINTER), X, OWN$SHIP(POINTER), Y);
  IF OS$LAST$POSI.FLAG
  THEN DO,
    IF TEMP
    THEN DO,
      CALL START$VECTOR$SOLID(OS$LAST$POSI.X, OS$LAST$POSI.Y);
      CALL NORMALIZE(OWN$SHIP(POINTER), X, OWN$SHIP(POINTER), Y,
                    .X, .Y);
      CALL STOP$VECTOR$SOLID(X, Y);
      CALL DRAW$OWN$SHIP$SYMBOL(X, Y);
      OS$LAST$POSI.X = X;
      OS$LAST$POSI.Y = Y;
    END;
  ELSE DO,
    OS$LAST$POSI.FLAG = FALSE;
  END;
END;
ELSE DO,
  IF TEMP
  THEN DO;

```

PLASMA\$MODULE

PLASMA\$OS

```
CALL NORMALIZE( OWN$SHIP<POINTER> . X , OWN$SHIP<POINTER> . Y ,  
               X , Y ) ;  
CALL DRAW$OWN$SHIP$SYMBOL( X , Y ) ;  
OS$LAST$POS1 . FLAG = TRUE ;  
OS$LAST$POS1 . X = X ;  
OS$LAST$POS1 . Y = Y ;  
END ;  
END ;  
END PLASMA$OS ;
```


PLASMA\$MODULE

DRAW\$EVERYTHING

```

/*****
*
* DRAW$EVERYTHING:
* THIS PROCEDURE IS USED TO REDRAW THE ENTIRE DISPLAY.
*
*****/
DRAW$EVERYTHING: PROCEDURE PUBLIC;
  DCL (POINTER, I, J, P, TEMP) BYTE;
  CALL CLEAR$PLASMA;
  POINTER = OWN$SHIP$INFO.POINTER;
  IF OWN$SHIP$INFO.FLAG
  THEN DO;
    P = POINTER + 1;
    DO I = 0 TO 29;
      IF P >= 30 THEN P = 0;
      OWN$SHIP$INFO.POINTER = P;
      CALL PLASMA$OS;
      P = P + 1;
    END;
  ELSE DO;
    DO I = 0 TO POINTER;
      OWN$SHIP$INFO.POINTER = I;
      CALL PLASMA$OS;
    END;
  END;
  OWN$SHIP$INFO.POINTER = POINTER;
  DO I = 0 TO 14;
    IF CONTACT$INFO(I).DESIG <> 00H
    THEN DO;
      POINTER = CONTACT$INFO(I).POINTER;

```

PLASMA\$MODULE

DRAW\$EVERYTHING

```

TEMP = CONTACT$INFO(I). POINTER / 15;
TEMP = (TEMP + 1) * 15;
IF CONTACT$INFO(I). FLAG
THEN DO;
    P = POINTER + 1;
    DO J = 0 TO 14;
        IF P >= TEMP THEN P = (POINTER / 15) * 15;
        CONTACT$INFO(I). POINTER = P;
        CALL PLASMA$CONTACT(I);
        P = P + 1;
    END;
END;
ELSE DO;
    DO J = ((POINTER / 15) * 15) * 15 TO POINTER;
        CONTACT$INFO(I). POINTER = J;
        CALL PLASMA$CONTACT(I);
    END;
END;
CONTACT$INFO(I). POINTER = POINTER;
END;
END DRAW$EVERYTHING;

```

PLASMA\$MODULE

DISPLAY\$PLASMA\$SCALE

```
/******  
* DISPLAY$PLASMA$SCALE:  
* THIS PROCEDURE IS USED TO DISPLAY AT THE PLASMA DISPLAY, THE SCALE BEING  
* USED TO DRAW THE PICTURE.  
*  
*****/  
DISPLAY$PLASMA$SCALE: PROCEDURE PUBLIC;  
  DCL TEMP BYTE;  
  TEMP = FP$FORMAT(.SYSTEM.SCALE, .BUFFER(7), 2, 2);  
  BUFFER(12), BUFFER(13) = EOL;  
  BUFFER(11) = BUFFER(10);  
  BUFFER(10) = BUFFER(9);  
  BUFFER(9) = POINT;  
  CALL PLASMA$PRINT$STRING(0, 2, .BUFFER);  
  END DISPLAY$PLASMA$SCALE;
```

PLASMA\$MODULE

REORIENT\$PS

```

/*****
*
* REORIENT$PS:
* THIS PROCEDURE IS USED TO DEFINE A NEW REFERENCE POINT FOR THE PICTURE
* TO BE PRESENTED AT THE PLASMA DISPLAY.
*
*****
*****
REORIENT$PS: PROCEDURE PUBLIC;
DCL DESIG ADDRESS,
DCL (TYPE, OK, REORIENT, INDEX, TEMP, FLAG, COUNT) BYTE;
DCL TITLE (*) BYTE DATA
    (
        MSG0 (*) BYTE DATA
            ('IF FIXED REORIENTATION IS DESIRED, TYPE 0. $$'),
        MSG1 (*) BYTE DATA
            ('IF OWN SHIP AT CENTER IS DESIRED, TYPE 1. $$'),
        MSG2 (*) BYTE DATA
            ('IF A CONTACT IS DESIRED AT CENTER, TYPE 2. $$'),
        MSG3 (*) BYTE DATA
            ('ENTER VALUE: $$'),
        MSG4 (*) BYTE DATA
            ('ENTER POINT DESIRED TO BE AT CENTER: $$'),
        MSG5 (*) BYTE DATA
            ('ENTER CONTACT DESIG AS REQUESTED: $$'),
        MSG6 (*) BYTE DATA
            ('DESIG NOT IN USE. $$');
    )
IF SYSTEM.NUMCTS > 0
THEN DO;
    FLAG = TRUE;
    COUNT = '2';

```


PLASMA\$MODULE

REORIENT\$PS

```

END;
ELSE DO;
  FLAG = FALSE;
  COUNT = '1';
  END;
CALL CLEAR$STRUCTURES;
CALL SET$WINDOW;
OK = 0;
DO WHILE OK = 0;
  CALL CRT$PRINT$STRING(. TITLE);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG0);
  CALL SEND$CRLF;
  CALL CRT$PRINT$STRING(. MSG1);
  CALL SEND$CRLF;
  IF FLAG
  THEN DO;
    CALL CRT$PRINT$STRING(. MSG2);
    CALL SEND$CRLF;
  END;
  CALL CRT$PRINT$STRING(. MSG3);
  REORIENT = CRT$READ;
  DO WHILE (REORIENT < '0') OR (REORIENT > COUNT);
    CALL SEND$BEL;
    REORIENT = CRT$READ;
  END;
  CALL CRT$WRITE(REORIENT);
  CALL SEND$CRLF;
  OK = CHECK$INPUT;
  CALL CLEAR$LOW$SCREEN;
END;

```

PLASMA\$MODULE

REORIENT\$PS

```

REORIENT = REORIENT - 30H;
DO CASE REORIENT;

DO; /* CASE 0. FIXED REORIENTATION. */
  OK = 0;
  DO WHILE OK = 0;
    CALL CRT$PRINT$STRING(.TITLE);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(.MSG4);
    TYPE = CRT$READ;
    DO WHILE (TYPE < '0') OR (TYPE > '7');
      CALL SEND$BEL;
      TYPE = CRT$READ;
    END;
    CALL CRT$WRITE(TYPE);
    CALL SEND$CRLF;
    OK = CHECK$INPUT;
    CALL CLEAR$LOW$SCREEN;
  END;
  CALL FIXED$REORIENTATION(TYPE - 30H);
END;
DO; /* CASE 1. OWN SHIP AT CENTER. */
  CALL PUT$OS$CENTER;
END;
DO; /* CASE 2. CONTACT AT CENTER. */
  OK = 0FFH;
  DO WHILE OK = 0FFH;
    CALL CRT$PRINT$STRING(.TITLE);
    CALL SEND$CRLF;
    CALL CRT$PRINT$STRING(.MSG5);
    CALL SEND$CRLF;
  
```

PLASMA\$MODULE

REORIENT\$PS

```

DESIG = GET$DESIG;
INDEX = CHECK$DESIG(DESIG);
IF INDEX = 0FFH
THEN DO;
    CALL CRT$PRINT$STRING(. MSG6);
    CALL SEND$CRLF;
    CALL SEND$CRLF;
    CALL CHECK$GO$KEY;
    END;
    OK = INDEX;
    CALL CLEAR$LOW$SCREEN;
    END;
    CALL PUT$CONTACT$CENTER(INDEX);
    END;
    END; /* END CASE */

CALL DRAW$EVERYTHING;
CALL DISPLAY$PLASMA$SCALE;
END REORIENT$PS;

END PLASMA$MODULE;

```

CRT

CRT

CRT: DO;

CRT\$WRITE:
PROCEDURE (A) EXTERNAL;
DECLARE A BYTE; END;

CRT\$PRINT\$STRING:
PROCEDURE (A) EXTERNAL;
DECLARE A ADDRESS; END;

DECLARE
LIT LITERALLY 'LITERALLY',
DCL LIT 'DECLARE';

DCL LF LIT '0AH',
MASTER\$CLEAR LIT '1EH',
BLINK LIT '0EH',
ETEOL LIT '17H',
EOL LIT '24H',
HOME LIT '02H',
TAB LIT '09H',
FS LIT '1CH',
PROT\$FIELD LIT '0FH',
END\$PF LIT '18H',
SPACE LIT '20H';

/* LINE FEED. */
/* MASTER CLEAR. */
/* BLINK ON. */
/* ERASE TO END OF LINE. */
/* END OF LINE. */
/* HOME. */
/* TAB. */
/* FORWARD CURSOR. NON-DESTRUCTIVE. */
/* START PROTECTED FIELD. */
/* STOP PROTECTED FIELD. */
/* SPACE. */

DCL LIN1 (*) BYTE DATA
('TIME: (\$3) \X5\X4\X5\X4LAST MARK2\4\X6\X5\X6C P A27# '),

CRT

CRT

```
LIN2A (*) BYTE DATA
      ('$2:$2:$2%4\DESIG\TYPE\CLASS\ TIME\ BRG \ RNG \#'),
LIN2B (*) BYTE DATA
      ('COURSE\SPEED\ TIME\ BRG \ RNG #'),
LIN3A (*) BYTE DATA
      ('%<#'),
LIN4A (*) BYTE DATA
      ('LAT:%8#'),
LIN5A (*) BYTE DATA
      ('$3:$2:$1 $1 #'),
LIN6A (*) BYTE DATA
      ('LONG:%7#'),
LIN9A (*) BYTE DATA
      ('COURSE:$3.$1#'),
LIN10A (*) BYTE DATA
      ('SPEED: $2.$1#'),
LIN12A (*) BYTE DATA
      ('CONTACTS: #'),
LIN13A (*) BYTE DATA
      (' $2F $2H $2U#'),
LIN15A (*) BYTE DATA
      ('MODE:$7#'),
LIN16 (*) BYTE DATA
      ('%XX#'),
LINF (*) BYTE DATA
      ('%5%4%5%5%6%6%5%5%5%6#'),
LINE (*) BYTE DATA
      (' $2 \ $2 \ $3 \ $2:$2:$3.$1\ $6\ $3.$1\ $2.$1\ $2:$2:$3.$1\ $6#'),
```

CRT

CRT\$MASTER\$CLEAR;

```
/******  
* CRT$MASTER$CLEAR:  
* THIS PROCEDURE WILL CLEAR THE ENTIRE SCREEN, AND PUT THE CURSOR AT HOME.  
*  
*****  
CRT$MASTER$CLEAR: PROCEDURE PUBLIC;  
  (1) : "111 CRT$WRITE(MASTER$CLEAR);  
      END CRT$MASTER$CLEAR;
```

CRT

SET\$LOW\$HOME

```

/*****
* SET$LOW$HOME:
* THIS PROCEDURE WILL LOCATE THE CURSOR AT THE FIRST COLUMN IN ROW 17.
*
*****/
SET$LOW$HOME: PROCEDURE PUBLIC;
  DCL I BYTE;
  CALL CRT$WRITE(HOME);
  DO I = 1 TO 8;
    CALL CRT$WRITE(LF);
  END;
END SET$LOW$HOME;

```

AD-A059 603

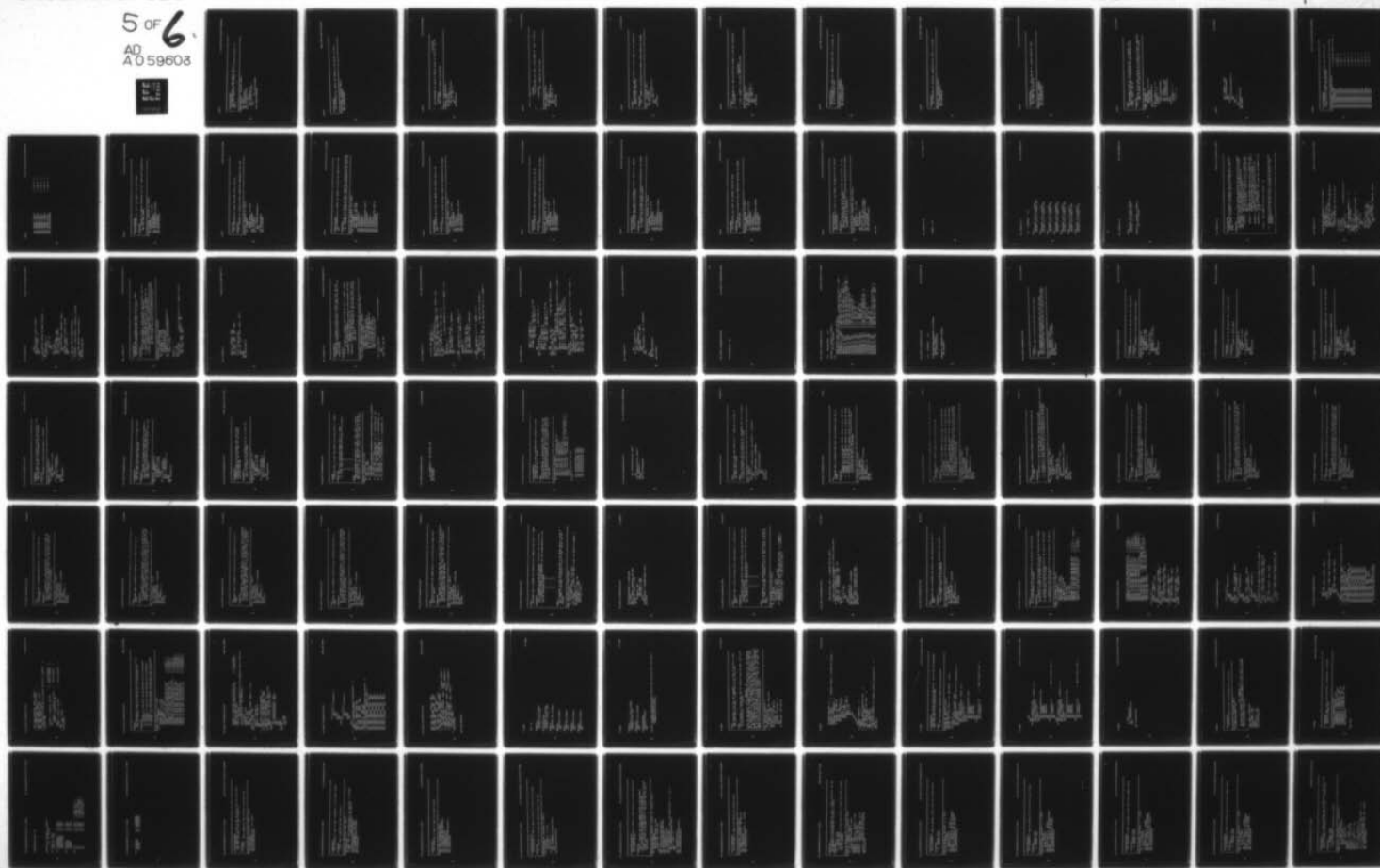
NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF
A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOT--ETC(U)
JUN 78 A L GONCALVES, J E CUBA BRAVO

F/G 17/7

UNCLASSIFIED

NL

5 OF 6
AD
A059603



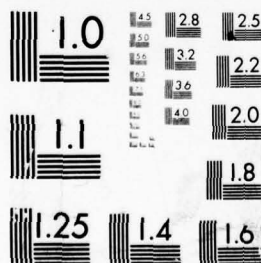
1 F 1 E D

5 OF

6

AD

A 0 5 9 6 0 3



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

CRT

CLEAR\$LOW\$SCREEN

```

/*****
*
* CLEAR$LOW$SCREEN:
*   THIS PROCEDURE WILL CLEAR ROWS 17 THRU 24.
*   AFTER THIS OPERATION, THE CURSOR WILL BE PLACED AT COLUMN 1 IN
*   ROW 17.
*
*****
CLEAR$LOW$SCREEN: PROCEDURE PUBLIC;
  DCL I BYTE;
  CALL SET$LOW$HOME;
  CALL CRT$WRITE(ETEOL);
  DO I = 1 TO 7;
    CALL CRT$WRITE(LF);
    CALL CRT$WRITE(ETEOL);
  END;
  CALL SET$LOW$HOME;
  END CLEAR$LOW$SCREEN;
*****/
```

SET\$HIGH\$HOME

CRT

```
/******  
* SET$HIGH$HOME:  
* THIS PROCEDURE WILL LOCATE THE CURSOR AT THE COLUMN 1 AT ROW 1.  
*  
*****  
SET$HIGH$HOME: PROCEDURE PUBLIC;  
CALL CRT$WRITE(HOME);  
END SET$HIGH$HOME;
```

CRT

PUT\$SPACE

```
/******  
*  
*   THIS PROCEDURE WILL WRITE A CERTAIN NUMBER OF SPACES AT THE CRT.  
*  
*   *  
*   * PARAMETERS:  
*   * - NUM. - NUMBER OF SPACES TO BE WRITTEN.  
*  
*   *  
*****  
PUT$SPACE: PROCEDURE(NUM);  
  DCL NUM BYTE;  
  DO WHILE NUM > 0;  
    CALL CRT$WRITE(SPACE);  
    NI JM = NUM - 1;  
  END;  
END PUT$SPACE;
```


CRT

PUT\$TAB

```

/***** *1 *****/
*
* PUT$TAB:
* THIS PROCEDURE WILL SEND A GIVEN NUMBER OF 'TABS' TO THE CRT.
*
* PARAMETERS:
*   * - NUM. - NUMBER OF TABS DESIRED.
*
*****/
PUT$TAB: PROCEDURE(NUM);
DCL NUM BYTE;
DO WHILE NUM > 0;
  CALL CRT$WRITE(TAB);
  NUM = NUM - 1;
END;
END PUT$TAB;

```

CRT

PUT\$FS

```

/*****
*
* PUT$FS:
* THIS PROCEDURE WILL SEND TO THE CRT A GIVEN NUMBER OF NON-DESTRUCTIVE
* FORWARD CURSOR CHARACTERS.
*
* PARAMETERS:
* - NUM. - NUMBER OF NON-DESTRUCTIVE FORWARD CURSOR CHARACTERS DESIRED.
*
*****/
PUT$FS: PROCEDURE (NUM);
  DCL I IN BYTE;
  DO WHILE NUM > 0;
    CALL CRT$WRITE(FS);
    NUM = NUM - 1;
  END;
END PUT$FS;
```

CRT

PUT\$LF

```

/*****
*
* PUT$LF:
* THIS PROCEDURE WILL SEND AN SPECIFIED NUMBER OF LINE FEED CHARACTERS
* TO THE CRT.
*
* - NUM. - NUMBER OF LINE FEED CHARACTERS DESIRED.
*
*****
PUT$LF: PROCEDURE(NUM);
  DCL NUM BYTE;
  DO WHILE NUM > 0;
    CALL CRT$WRITE(LF);
    NUM = NUM - 1;
  END;
END PUT$LF;
*****/
```

CRT

START\$PROT\$FIELD

```

/*****
*
* START$PROT$FIELD:
* THIS PROCEDURE WILL CAUSE ALL CHARACTERS SENT AFTER IT FINISHES, TO
* BE IN A PROTECTED FIELD.
*
*****/
START$PROT$FIELD: PROCEDURE;
  CALL CRT$WRITE<PROT$FIELD>;
  END START$PROT$FIELD;
```


CRT

START\$BLINK

```
/******  
* START$BLINK:  
* THIS PROCEDURE WILL CAUSE ALL CHARACTERS SENT AFTER IT TO BLINK.  
*  
*****/  
START$BLINK: PROCEDURE PUBLIC;  
CALL CRT$WRITE(BLINK);  
END START$BLINK;
```

CRT

STOP\$PROT\$FIELD

```
/*  
*****  
* STOP$PROT$FIELD:  
* THIS PROCEDURE WILL CAUSE THE END TO 'INSERT PROTECTED FIELD',  
* 'ROLL MODE' AND 'BLINK ON'.  
*  
*****  
STOP$PROT$FIELD: PROCEDURE;  
CALL CRT$WRITE(END$PF);  
END STOP$PROT$FIELD;
```

CRT

INTERP

```
*****
* INTERP:
* THIS PROCEDURE IS USED BY 'INIT$HIGH$SCREEN' TO INTERPRET THE
* STREAM OF CHARACTERS USED TO INITIALIZE THE HIGH PORTION OF THE
* SCREEN. (ROWS 1 THRU 16)
*
* PARAMETERS:
* - A - POINTER TO THE FIRST LOCATION OF A STRING OF CONTROL CHARACTERS
* USED TO FORMAT THE ROWS ON THE HIGH PORTION OF THE SCREEN.
*
*****
INTERP: PROCEDURE (A);
  DCL A ADDRESS,
  ARRAY BASED A (80) BYTE,
  (I, NUM) BYTE;
  CALL START$PROT$FIELD;
  I = 0;
  DO WHILE ARRAY(I) <> '#';
    IF ARRAY(I) = '$'
      THEN DO;
        CALL STOP$PROT$FIELD;
        I = I + 1;
        NUM = ARRAY(I) - 30H;
        CALL PUT$SPACE(NUM);
        CALL START$PROT$FIELD;
      END;
    ELSE DO;
      IF ARRAY(I) = '%'
        THEN DO;
          I = I + 1;
        END;
    END;
  END;
*****
```

INTERP

CRT

```
NUM = ARRAY(1) - 30H;  
CALL PUT$SPACE(NUM);  
END;  
ELSE DO;  
CALL CRT$WRITE(ARRAY(1));  
END;  
END;  
I = I + 1;  
END;  
CALL STOP$PROT$FIELD;  
END INTERP;
```


NEEDS OF THE PEOPLE

395

CRT

INIT\$HIGH\$SCREEN

```
CALL INTERP(. LIN12A);  
CALL INTERP(. LINE);  
CALL INTERP(. LIN13A);  
CALL INTERP(. LINF);  
CALL INTERP(. LIN3A);  
CALL INTERP(. LINE);  
CALL INTERP(. LIN15A);  
CALL INTERP(. LINF);  
CALL INTERP(. LIN16);  
END INIT$HIGH$SCREEN;
```

```
/* LINE 12. */  
/* LINE 13. */  
/* LINE 14. */  
/* LINE 15. */  
/* LINE 16. */
```

CRT

PRINT\$TIME\$ZONE

```

/*****
*
* PRINT$TIME$ZONE:
* THIS PROCEDURE WILL PRINT THE CURRENT TIME ZONE NUMBER.
*
* PARAMETERS:
* - A - POINTER TO A STRING CONTAINING THE ASCII CHARACTERS REPRESENTING
* THE TIME ZONE NUMBER.
*
*****/
PRINT$TIME$ZONE: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        BUFFER BASED A (5) BYTE;
  CALL SET$HIGH$HOME;
  CALL CRT$PRINT$STRING(. BUFFER);
  CALL SET$LOW$HOME;
  END PRINT$TIME$ZONE;

```

CRT

PRINT\$TIME

```

/*****
*
* PRINT$TIME:
* THIS PROCEDURE WILL PRINT THE LOCAL TIME AT THE CRT.
*
* PARAMETERS:
* - A - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE TIME.
*
*****/
PRINT$TIME: PROCEDURE (A) PUBLIC;
DCL A ADDRESS,
      I BYTE,
      BUFFER BASED A (8) BYTE;
CALL SET$HIGH$HOME;
CALL PUT$TAB(1);
DO I = 0 TO 5;
    CALL CRT$WRITE(BUFFER(I));
END;
CALL SET$LOW$HOME;
END PRINT$TIME;

```


CRT

PRINT\$LAT\$LONG

```

/*****
*
* PRINT$LAT$LONG:
* THIS PROCEDURE WILL DISPLAY THE CURRENT LATITUDE AND LONGITUDE AT THE
* CRT.
*
* PARAMETERS:
* - A - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE LATITUDE.
* - B - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE LONGITUDE.
*
*****/
PRINT$LAT$LONG: PROCEDURE (A,B) PUBLIC;
  DCL (A,B) ADDRESS,
        BUFF1 BASED A (10) BYTE,
        BUFF2 BASED B (10) BYTE;
  CALL SET$HIGH$HOME;
  CALL PUT$LF(2);
  CALL PUT$TAB(16);
  CALL PUT$FS(6);
  CALL CRT$PRINT$STRING(. BUFF1);
  CALL SET$HIGH$HOME;
  CALL PUT$LF(3);
  CALL PUT$TAB(16);
  CALL PUT$FS(6);
  CALL CRT$PRINT$STRING(. BUFF2);
  CALL SET$LOW$HOME;
  END PRINT$LAT$LONG;

```

CRT

PRINT\$COURSE

```

/*****
*
* PRINT$COURSE:
* THIS PROCEDURE WILL PRINT THE CURRENT OWN COURSE VALUE AT THE
* CRT.
*
* PARAMETERS:
* - A - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE COURSE.
*
*****/
PRINT$COURSE: PROCEDURE (A) PUBLIC;
DCL A ADDRESS;
      BUFFER BASED A (6) BYTE;
      CALL SET$HIGH$HOME;
      CALL PUT$LF(4);
      CALL PUT$TAB(17);
      CALL CRT$PRINT$STRING(.BUFFER);
      CALL SET$LOW$HOME;
      END PRINT$COURSE;

```

CRT

PRINT\$SPEED

```

/*****
*
* PRINT$SPEED:
* THIS PROCEDURE WILL PRINT THE CURRENT OWN SPEED AT THE CRT.
*
* PARAMETERS:
* - A - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE SPEED.
*
*****/
PRINT$SPEED: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        BUFFER BASED A (5) BYTE;
  CALL SET$HIGH$HOME;
  CALL PUT$LF(5);
  CALL CRT$PRINT$STRING(. BUFFER);
  CALL SET$LOW$HOME;
  END PRINT$SPEED;

```

CRT

PRINT\$CONTACTS

```

/*****
*
* PRINT$CONTACTS:
* THIS PROCEDURE WILL PRINT THE NUMBER OF FRIEND, HOSTILE AND UNKNOWN
* CONTACTS BEING PROCESSED BY THE SYSTEM.
*
* PARAMETERS:
* - A - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE NUMBER
*   OF FRIEND, HOSTILE AND UNKNOWN CONTACTS.
*
*****/
PRINT$CONTACTS: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        BUFFER BASED A (8) BYTE;
  CALL SET$HIGH$HOME;
  CALL PUT$LF(6);
  CALL PUT$TAB(17);
  CALL CRT$PRINT$STRING(. BUFFER);
  CALL SET$LOW$HOME;
  END PRINT$CONTACTS;

```


CRT

PRINT\$MODE

```

/*****
*
* PRINT$MODE:
* THIS PROCEDURE WILL PRINT THE CURRENT OPERATING MODE.
*
* PARAMETERS:
* - A - POINTER TO A STRING OF ASCII CHARACTERS DEFINING THE CURRENT
* OPERATING MODE.
*
*****/
PRINT$MODE: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
        BUFFER BASED A (9) BYTE;
  CALL SET$HIGH$HOME;
  CALL PUT$LF(7);
  CALL PUT$TAB(17);
  CALL CRT$PRINT$STRING(.BUFFER);
  CALL SET$LOW$HOME;
  END PRINT$MODE;
/
```

CRT

PRINT\$CONTACT\$INFO

```

/*****
*
* PRINT$CONTACT$INFO:
* THIS PROCEDURE WILL PRINT ALL THE CURRENT INFORMATION OF ANY CONTACT
* BEING DISPLAYED.
*
* PARAMETERS:
* - NUM - REPRESENTS THE RELATIVE POSITION FROM TOP TO BOTTOM, OF THE
* CONTACT LINE DESIRED TO BE UPDATED.
* - A - POINTER TO A STRING OF ASCII CHARACTERS REPRESENTING THE CONTACT
* INFORMATION TO BE DISPLAYED.
*
* *****/
PRINT$CONTACT$INFO: PROCEDURE (NUM,A) PUBLIC;
  DCL A ADDRESS,
       BUFFER BASED A (44) BYTE,
       NUM BYTE;
  CALL SET$HIGH$HOME;
  CALL PUT$LF(NUM + 1);
  IF NUM = 4 THEN CALL PUT$TAB(2);
  CALL CRT$PRINT$STRING(.BUFFER);
  CALL SET$LOW$HOME;
  END PRINT$CONTACT$INFO;

END CRT;
```

FLTASCI I

FLTASCI I

FLTASCI I: DO

FLTASCI I

EXTERNALS

/*** EXTERNALS: ***/

EDIV:

PROCEDURE (A,B,C,D) EXTERNAL;
DECLARE (A,B,C,D) ADDRESS; END;

FMUL:

PROCEDURE (A,B,C) EXTERNAL;
DECLARE (A,B,C) ADDRESS; END;

FDIV:

PROCEDURE (A,B,C) EXTERNAL;
DECLARE (A,B,C) ADDRESS; END;

FADD:

PROCEDURE (A,B,C) EXTERNAL;
DECLARE (A,B,C) ADDRESS; END;

FSUB:

PROCEDURE (A,B,C) EXTERNAL;
DECLARE (A,B,C) ADDRESS; END;

FLTDS:

PROCEDURE (A,B) EXTERNAL;
DECLARE (A,B) ADDRESS; END;

FIXSD:

PROCEDURE (A,B) EXTERNAL;
DECLARE (A,B) ADDRESS; END;

FLTASCI I

EXTERNALS

```
FZTST:
  PROCEDURE (A,B) BYTE EXTERNAL;
  DECLARE (A,B) ADDRESS; END;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';
```

FLTASCII

ASCII TO FLOAT

```

/*****
*
* ASCII TO FLOAT:
*   PROCEDURE USED TO CONVERT A STRING OF ASCII CHARACTERS INTO A FLOATING
*   POINT NUMBER, ACCORDING TO THE FORMAT REQUIRED TO OPERATE ON THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - A - POINTER TO A N BYTE VECTOR CONTAINING THE ASCII STRING OF NUMBERS
*     REPRESENTING A DECIMAL VALUE DESIRED TO BE REPRESENTED IN FLOATING
*     POINT FORMAT. IF THE N BYTE VECTOR IS REPRESENTED BY THE
*     NAME 'BUFFER', THEN THE FOLLOWING RULES APPLY:
*     BUFFER(0) - CONTAINS THE DECIMAL NUMBER OF ASCII CHARACTERS PRE-
*     SENT IN THE BUFFER. A VALUE OF 0 WILL DENOTE THE NUM-
*     BER 0.0.
*     BUFFER(1) - CONTAINS THE DECIMAL NUMBER OF ASCII CHARACTERS THAT
*     REPRESENT THE INTEGER PORTION OF THE NUMBER. A VALUE OF
*     0 WILL MEAN THAT THE NUMBER IS LESS THAN ONE.
*     BUFFER(2) --> BUFFER(N-2). - EACH BYTE CONTAINS AN ASCII CHA-
*     RACTER. (HEXADECIMAL VALUE)
*     BUFFER(N-1). - A VALUE OF DECIMAL 1 WILL MEAN THAT THE NUMBER IS NE-
*     GATIVE.
*
*   - LIMIT. - TOTAL NUMBER OF BYTES IN 'BUFFER'. (N)
*
*   - B. - POINTER TO A FOUR BYTE VECTOR THAT WILL CONTAIN THE FLOATING POINT
*     REPRESENTATION OF THE ASCII STRING CONTAINED IN BUFFER(2) THRU
*     BUFFER(8).
*
*****/

```

FLTASCII

ASCII\$TO\$FLOAT

```

ASCII$TO$FLOAT:PROCEDURE(A,LIMIT,B) PUBLIC;
DCL (A,B) ADDRESS,
    BUFFER BASED A BYTE,
    VECTOR BASED B (4) BYTE,
    TEN$FLOAT (4) BYTE DATA (00H,00H,20H,41H),
    TEMP (4) BYTE,
    RESULT (4) BYTE,
    (I,NUMINT,NUMDEC,NUM,LIMIT,T0,T1) BYTE;

T0 = BUFFER;
A = A + 1;
T1 = BUFFER;
DO I = 0 TO 3;
    VECTOR(I) = 0;
END;
/* INITIALIZE VECTOR */

/* CHECK IF PROPOSED NUMBER IS 0. */
IF T0 = 0 THEN RETURN;
NUMINT = T1;
NUMDEC = T0 - T1;
/* FIND INTEGER PORTION OF NUMBER */
DO WHILE NUMINT > 0;
    DO I = 0 TO 3;
        TEMP(I) = 0;
    END;
    A = A + 1;
    TEMP(0) = BUFFER - 30H;
    CALL FLTDC(TEMP, RESULT);
    IF NUMINT > 1 THEN DO;
        DO I = 1 TO NUMINT - 1;
            CALL FMUL(RESULT, TEN$FLOAT, RESULT);
        END;
    END;

```

FLTASCII

ASCII\$TO\$FLOAT

```

END;
CALL FADD(.RESULT,.VECTOR,.VECTOR);
NUMINT = NUMINT - 1;
END;

/* FIND DECIMAL PORTION OF NUMBER PROPOSED. */
NUM = 0;
DO WHILE NUMDEC > 0;
DO I = 0 TO 3;
TEMP(I) = 0;
END;
A = A + 1;
NUM = NUM + 1;
TEMP(0) = BUFFER - 30H;
CALL FLTDS(.TEMP,.RESULT);
DO I = 1 TO NUM;
CALL FDIV(.RESULT,.TEN$FLOAT,.RESULT);
END;
CALL FADD(.RESULT,.VECTOR,.VECTOR);
NUMDEC = NUMDEC - 1;
END;

/* CHECK FOR SIGN OF NUMBER PROPOSED. */
A = A + 1;
IF BUFFER = 1 THEN VECTOR(3) = VECTOR(3) XOR 080H;

/* ALL DONE. RETURN TO CALLING MODULE. */
RETURN;
END ASCII$TO$FLOAT;

```


FLTASCII

FRAC\$TO\$ASCII

```

/*****
*
* FRAC$TO$ASCII:
*   PROCEDURE USED TO CONVERT A FRACTIONAL PART OF A F.P. NUMBER
*   INTO AN ASCII REPRESENTATION AND STORE IT IN A BUFFER.
*
* PARAMETERS:
*   - A - POINTER TO A VARIABLE - WHICH CONTAINS THE FRACTIONAL PART OF A
*     F.P. NUMBER- PASSED BY FLOAT$TO$ASCII PROCEDURE.
*   - B - POINTER TO A BUFFER PARTIALLY FILLED W/ THE INTEGER PART OF
*     A F.P. NUMBER CONVERTED TO ASCII BY FLOAT$TO$ASCII.
*   - DEC$POS - POINTER TO A VARIABLE WHICH INDICATES THE POSITION OF DECIMAL
*     POINT TO BE SET ; IT RETURN THE ADDRESS WHICH CONTAINS
*     THE FIRST VACANT POSITION IN THE BUFFER AFTER FILLED.
*
*****/
FRAC$TO$ASCII: PROCEDURE (A,B,DEC$POS);
  DCL (A,B,DEC$POS) ADDRESS;
  DCL TERM BASED A (4) BYTE,
      BUFF BASED B (28) BYTE,
      C BASED DEC$POS BYTE;
  DCL TEN$FLOAT (4) BYTE DATA (00H,00H,20H,41H);
  DCL TEMP (4) BYTE, TEMP1 BYTE DATA (4),
      (FLAG, I) BYTE;

  BUFF(C) = ' ';
  C = C + 1;
  FLAG = 0FFH;
  DO WHILE FLAG;
    CALL FMUL (,TERM, ,TEN$FLOAT, ,TERM);
    CALL FIXSD (,TERM, ,TEMP);
    /* SET UP FRAC PART OF F.P. NUMBER TO ASCII */

```

FRAC\$TO\$ASCII

FLTASCII

```
      BUFF(C) = TEMP(0) + 30H;  
      CALL FLTDS (.TEMP, .TEMP);  
      CALL FSUB (.TERM, .TEMP, .TERM);  
      C = C + 1;  
      IF C = 26 THEN RETURN;  
      FLAG = NOT (FLAG = FZTST (.TERM, .TEMP1));  
      END;  
      RETURN;  
      END FRAC$TO$ASCII;
```

FLTASCII

FLAT\$TO\$ASCII

```

/*****
*
* FLAT$TO$ASCII:
* PROCEDURE USED TO CONVERT A FLOATING POINT NUMBER -SEE INTEL SBC 310-
* INTO AN ASCII REPRESENTATION AND STORE IT IN A BUFFER TO BE SENT TO THE
* CALLING MODULE.
*
* PARAMETERS:
* - FLOAT -POINTER TO A VARIABLE -WHICH CONTAINS THE F.P. NUMBER- PASSED
* BY THE CALLING MODULE;
* - ASC$BUFFER -POINTER TO A BUFFER W/ LENGTH 28 WHICH CONTAINS THE ASCII
* REPRESENTATION OF THE F.P. NO.;
* - END$BUFFER - POINTER TO VARIABLE PASSED TO THE CALLING MODULE INDICA-
* TING THE FIRST VACANT BYTE OF BUFFER AFTER LAST PRINTABLE
* ASCII BYTE.
*
*****/
FLAT$TO$ASCII: PROCEDURE (FLOAT,ASC$BUFFER,END$BUFFER) PUBLIC;
DCL (FLOAT,ASC$BUFFER,END$BUFFER) ADDRESS,
M BASED FLOAT (4) BYTE,
A BASED ASC$BUFFER (28) BYTE,
DEC$POINT BASED END$BUFFER BYTE,
ONE (4) BYTE DATA (00H,00H,80H,3FH),
TEN (2) BYTE DATA (0AH,00H),
(TEMP,TEMP2,FRAC,REM) (4) BYTE,
(TEMP1,I,SIGN,EXP,FLAG) BYTE;

DO I = 0 TO LAST(A);
  A(I) = ' ';
END;
SIGN = M(3) AND 80H;

```

FLTASCII

FLTASCII

```

IF SIGN = 80H THEN A(0) = '-'
ELSE A(0) = ' '
EXP = SHL(M(3),1) OR SHR(M(2),7)
TEMP(3) = M(3) TEMP(2) = M(2) TEMP(1) = M(1) TEMP(0) = M(0)
IF (EXP < 7FH) AND (EXP >= 00H) THEN I = 0
IF EXP = 7FH THEN I = 1
IF (EXP <= 0FEH) AND (EXP > 7FH) THEN I = 2
DO CASE I
DO
/* CASE 0: APPLIED FOR -1.0 < F.P. NUMBERS < 1.0 */
DEC$POINT = 2
A(1) = '0'
TEMP(3) = TEMP(3) AND 7FH
CALL FRAC$TO$ASCII (.TEMP, A, DEC$POINT)
END
DO
/* CASE 1: APPLIED FOR 2.0 > F.P. NUMBER >= 1.0
OR -2.0 < F.P. NUMBER <= -1.0 */
DEC$POINT = 2
A(1) = '1'
TEMP(3) = TEMP(3) AND 7FH
CALL FSUB (.TEMP, ONE, TEMP)
CALL FRAC$TO$ASCII (.TEMP, A, DEC$POINT)
END
DO
/* CASE 2: APPLIED FOR -1.0 > F.P. NUMBER > 1.0 */
DEC$POINT = 1
EXP = EXP - 7FH
DO I = 1 TO 23 - EXP
/* TAKE OUT BIAS OF EXPONENT */
/* NEXT 2 INTERACTIVES "DO" TAKE OUT FRACTIONAL PART OF F.P. NUMBER */

```


FLTASCI I

FL0AT\$T0\$FASCI I

```

TEMP(2) = SCR(TEMP(2),1);
TEMP(1) = SCR(TEMP(1),1);
IF CARRY THEN TEMP(0) = SHR(TEMP(0),1) OR 80H;
    ELSE TEMP(0) = SHR(TEMP(0),1);
END;
EXP = EXP OR EXP; /* RESET CARRY BIT */
DO I = 1 TO 23 -EXP;
    TEMP(0) = SCL(TEMP(0),1);
    TEMP(1) = SCL(TEMP(1),1);
    IF CARRY THEN TEMP(2) = SHL(TEMP(2),1) OR 01H;
        ELSE TEMP(2) = SHL(TEMP(2),1);
    END;
CALL FSUB (M, TEMP, FRAC); /* SAVE FRAC PART OF F.P. NUMBER */
FRAC(3) = FRAC(3) AND 7FH;
CALL FIXSD (TEMP, TEMP);
IF (TEMP1 := TEMP(3) AND 80H) = 80H THEN
    DO; /* PERFORMS TWO'S COMPLEMENT OF A NEGATIVE F.P. NUMBER */
        TEMP(0) = NOT TEMP(0); TEMP(1) = NOT TEMP(1);
        TEMP(2) = NOT TEMP(2); TEMP(3) = NOT TEMP(3);
        TEMP(0) = TEMP(0) + 1; TEMP(1) = TEMP(1) PLUS 0;
        TEMP(2) = TEMP(2) PLUS 0; TEMP(3) = TEMP(3) PLUS 0;
    END;
FLAG = 0FFH;
DO WHILE FLAG; /* SET UP INTEGER PART OF F.P. NUMBER TO ASCII */
    CALL EDIV (TEMP, TEN, TEMP, REM);
    AC(DEC$POINT) = REM(0) + 30H;
    DEC$POINT = DEC$POINT + 1;
    CALL FLTDS (TEMP, TEMP);
    TEMP1 = 4;
    FLAG = NOT (FLAG := FZTST (TEMP, TEMP1));
    CALL FIXSD (TEMP, TEMP);

```

FL0AT\$T0\$ASCII

FLTASCII

```
END;
DO I = 1 TO DEC$POINT/2;
  TEMP1 = A(I);
  A(I) = A(DEC$POINT - I);
  A(DEC$POINT - I) = TEMP1;
END;
CALL FRAC$T0$ASCII (.FRAC., A., DEC$POINT);
END;

END;
RETURN;
END FL0AT$T0$ASCII;

END FLTASCII;
```

FLOATING\$POINT

FLOATING\$POINT

FLOATING\$POINT: DO;

FLOATING\$POINT

DECLARATIONS

```

/**** DECLARATIONS: ****/

DECLARE LIT LITERALLY 'LITERALLY',
      DCL LIT 'DECLARE',

DCL
  MEBAS ADDRESS PUBLIC DATA (0F790H) ,
  RES$TABLE (8) BYTE PUBLIC AT (0F790H), /* MEMORY RESERVED FOR F.P. BOARD. */
  OUT$OP$CODE LIT '010H', /* BASE OUTPUT PORT FOR F.P. BOARD. */
  IN$STATUS LIT '011H', /* INPUT PORT FOR STATUS OF F.P. BOARD. */
  IN$FLAG LIT '017H', /* INPUT PORT FOR FLAG FROM F.P. BOARD. */
  MEM$LOW LIT '011H', /* OUTPUT PORT FOR LOW PORTION OF MEMORY BASE. */
  MEM$HIGH LIT '012H', /* OUTPUT PORT FOR HIGH PORTION OF MEMORY BASE. */
  MUL$CODE LIT '00H', /* FIXED POINT MULTIPLICATION CODE. */
  DIV$CODE LIT '01H', /* FIXED POINT DIVISION CODE. */
  FMUL$CODE LIT '02H', /* F.P. MULTIPLICATION CODE. */
  FDIV$CODE LIT '03H', /* F.P. DIVISION CODE. */
  FADD$CODE LIT '04H', /* F.P. ADD CODE. */
  FSUB$CODE LIT '05H', /* F.P. SUBTRACT CODE. */
  FSQR$CODE LIT '06H', /* F.P. SQUARE CODE. */
  FSQRT$CODE LIT '07H', /* F.P. SQUARE ROOT CODE. */
  FLTDS$CODE LIT '08H', /* FIXED-TO-FLOAT CONVERSION CODE. */
  FIXSD$CODE LIT '09H', /* FLOAT-TO-FIXED CONVERSION CODE. */
  FCMPR$CODE LIT '0AH', /* F.P. COMPARE CODE. */
  FZTST$CODE LIT '0BH', /* F.P. TEST CODE. */
  EXCH$CODE LIT '0FH', /* EXCHANGE CODE. */
  EDIV$CODE LIT '0EH', /* EXTENDED FIXED POINT DIVISION CODE. */
  BUSY$MASK LIT '01H', /* MASK FOR BUSY SIGNAL FROM F.P. BOARD. */
  ERROR$MASK LIT '04H', /* MASK FOR ERROR CODE FROM F.P. BOARD.

```


FLOATING\$POINT

EXTERNALS

/*** EXTERNALS: ***/

CRT\$PRINT\$STRING:PROCEDURE (A) EXTERNAL;
 DECLARE A ADDRESS;
 END CRT\$PRINT\$STRING;

SEND\$CRLF: PROCEDURE EXTERNAL;
 END SEND\$CRLF;

FLOATING\$POINT

INIT\$FP

```

/*****
*
* INIT$FP:
*   PROCEDURE USED TO INITIALIZE THE FLOATING POINT MODULE.
*
* USAGE:
*   THIS PROCEDURE SHOULD BE CALLED ONE TIME FROM THE USER'S
*   MAIN PROGRAM BEFORE ATTEMPTING TO USE ANY OF THE ROUTINES PROVI-
*   DED FOR FLOATING POINT OPERATIONS WITH THE FLOATING POINT BOARD.
*
*****/
INIT$FP: PROCEDURE PUBLIC;
  OUTPUT(MEM$LOW) = LOW(MEBAS);
  OUTPUT(MEM$HIGH) = HIGH(MEBAS);
  RETURN;
END INIT$FP;

```

FLOATING\$POINT

ADJUST\$OP

```

/*****
*
* ADJUST$OP:
* PROCEDURE USED TO PUT TWO VECTORS INTO THE TABLE VECTOR.
*
* PARAMETERS:
* - A, B - POINTERS TO FIRST AND SECOND VECTOR VALUES.
*
*****/
ADJUST$OP: PROCEDURE(A, B);
    DCL (A, B) ADDRESS,
        OP1 BASED A (4) BYTE,
        OP2 BASED B (4) BYTE,
        I BYTE;
    DO I = 0 TO LAST(OP1);
        RES$TABLE(I) = OP1(I);
        RES$TABLE(I + 4) = OP2(I);
    END;
    RETURN;
END ADJUST$OP;

```

FLOATING\$POINT

ADJUST1\$OP

```

/*****
*
* ADJUST1$OP:
* PROCEDURE USED TO PUT ONE VECTOR INTO THE TABLE VECTOR.
*
* PARAMETERS:
* - A - POINTER TO VECTOR VALUE.
*
*****/
ADJUST1$OP: PROCEDURE (A);
DCL A ADDRESS,
      OP1 BASED A (4) BYTE,
      I BYTE;
DO I = 0 TO LAST(OP1);
  RES$TABLE(I) = OP1(I);
END;
RETURN;
END ADJUST1$OP;

```


FLOATING\$POINT

ADJUST2\$OP

```

/*****
*
* ADJUST2$OP:
* PROCEDURE USED TO PUT TWO ADDRESS VALUES INTO THE TABLE VECTOR.
*
* PARAMETERS:
* - A,B - POINTERS TO TWO ADDRESS VALUES.
*
*****/
ADJUST2$OP: PROCEDURE (A,B);
  DCL (A,B) ADDRESS,
    OP1 BASED A (2) BYTE,
    OP2 BASED B (2) BYTE,
    I BYTE;
  DO I = 0 TO LAST(OP1);
    RES$TABLE(I) = OP1(I);
    RES$TABLE(I + 4) = OP2(I);
  END;
RETURN;
END ADJUST2$OP;

```

FLOATING\$POINT

VAL\$RESULT

```

/*****
* VAL$RESULT:
* PROCEDURE USED TO GET THE RESULT IN FIRST FOUR BYTES OF THE
* TABLE VECTOR, AND PUT IT INTO ANOTHER VECTOR PROVIDED.
*
* PARAMETERS:
* - C - POINTER TO VECTOR ON WHICH RESULT IS DESIRED.
*
*****/
VAL$RESULT: PROCEDURE(C);
  DCL C ADDRESS;
  RESULT BASED C (4) BYTE,
  1 BYTE;
  DO I = 0 TO LAST(RESULT);
    RESULT(I) = RES$TABLE(I);
  END;
  RETURN;
END VAL$RESULT;

```

FLOATING\$POINT

VAL\$RESULT\$1

```

/*****
*
* VAL$RESULT$1:
* PROCEDURE USED TO PUT ALL EIGHT BYTES OF THE TABLE VECTOR
* INTO TWO FOUR BYTES VECTORS PROVIDED.
*
* PARAMETERS:
* - A,B - POINTERS TO VECTORS ON WHICH RESULT IS DESIRED. A
* MUST POINT TO FIRST VECTOR (FIRST FOUR BYTES OF
* TABLE), AND B MUST POINT TO THE SECOND VECTOR (LAST FOUR
* BYTES IN TABLE).
*
*****/
VAL$RESULT$1: PROCEDURE(A,B);
  DCL (A,B) ADDRESS,
    OP1 BASED B (4) BYTE,
    OP2 BASED A (4) BYTE,
    I BYTE;
  DO I = 0 TO LAST(OP1);
    OP2(I) = RES$TABLE(I);
    OP1(I) = RES$TABLE(I + 4);
  END;
RETURN;
END VAL$RESULT$1;
```

FLOATING\$POINT

VAL\$RESULT\$2

```
/******  
* VAL$RESULT$2:  
* PROCEDURE USED TO GET THE RESULT FROM FIXED POINT  
* DIVISION OPERATION, AND PUT THEM INTO TWO ADDRESS  
* LOCATIONS PROVIDED.  
*  
* PARAMETERS:  
* - C,R - POINTERS TO TWO ADDRESS LOCATIONS IN WHICH THE  
* RESULT IS DESIRED TO BE PLACED.  
*  
*****/  
VAL$RESULT$2: PROCEDURE (C,R);  
  DCL (C,R) ADDRESS,  
  OP1 BASED C (2) BYTE,  
  OP2 BASED R (2) BYTE,  
  I BYTE;  
  DO I = 0 TO LAST(OP1);  
    OP1(I) = RES$TABLE(I);  
    OP2(I) = RES$TABLE(I + 4);  
  END;  
END VAL$RESULT$2;
```


FLOATING\$POINT

COMPARE

```

/*****
*
* COMPARE:
* PROCEDURE USED TO CHECK FOR OUTPUT CONDITIONS FROM F. P. BOARD.
*
* PARAMETERS:
* - A - BYTE VALUE CONTAINING RESULT FROM F. P. BOARD.
* - B - BYTE VALUE CONTAINING CONDITION DESIRED TO BE CHECKED:
*   < ..... 0
*   <= ..... 1
*   > ..... 2
*   >= ..... 3
*   = ..... 4
*   <> ..... 5
*
* USAGE:
* TYPED PROCEDURE THAT RETURNS A 'TRUE' VALUE (001H) IF OUTPUT
* CONDITION FROM F. P. BOARD AND CONDITION DESIRED TO BE TESTED
* ARE SIMILAR. IF NOT SIMILAR, A 'FALSE' VALUE (000H) IS RETURNED.
*
*****/
COMPARE: PROCEDURE (A,B) BYTE;
    DCL TRUE LIT '01H',
    FALSE LIT '00H';
    DCL (A,B) BYTE,
    (LESS$THAN, LESS$OR$EQUAL, GREATER$THAN, GREAT$OR$EQUAL, EQUAL, NOT$EQUAL)
    BYTE DATA (0,1,2,3,4,5);
    IF ((A = 80H) AND (B = LESS$OR$EQUAL) OR (B = GREAT$OR$EQUAL) OR
        (B = EQUAL)) THEN RETURN TRUE;
    IF ((A = 40H) AND (B = GREATER$THAN) OR (B = GREAT$OR$EQUAL) OR
        (B = NOT$EQUAL)) THEN RETURN TRUE;
    IF ((A = 20H) AND (B = LESS$THAN) OR (B = LESS$OR$EQUAL) OR

```

FLOATING\$POINT

COMPARE

```
(B = NOT$EQUAL)) THEN RETURN TRUE;  
RETURN FALSE;  
END COMPARE;
```

FLOATING\$POINT

Float\$MSG\$error

```

/*****
*
* Float$MSG$error:
* PROCEDURE USED TO SEND A MESSAGE ERROR ACCORDING TO ERROR
* CODE PROVIDED BY F. P. BOARD.
*
* USAGE:
* UNTYPED PROCEDURE. IF CONDITION OF ERROR IS DETECTED, THIS
* PROCEDURE MUST BE CALLED IN ORDER TO OBTAIN ERROR CODE AND
* DISPLAY AN APPROPRIATE MESSAGE. NOTE THAT PROGRAM EXECUTION
* WILL BE STOPPED AND INTERRUPTS WILL BE ENABLED IF AN ERROR
* IS DETECTED.
*
*****/
Float$MSG$error: PROCEDURE;
    DCL MSG1(*) BYTE DATA ('DIVISION BY ZERO. $$$'),
        MSG2(*) BYTE DATA ('DOMAIN ERROR. $$$'),
        MSG3(*) BYTE DATA ('OVERFLOW. $$$'),
        MSG4(*) BYTE DATA ('UNDERFLOW. $$$'),
        MSG5(*) BYTE DATA ('INVALID FORMAT FOR FIRST ARGUMENT. $$$'),
        MSG6(*) BYTE DATA ('INVALID FORMAT FOR SECOND ARGUMENT. $$$'),
        ERROR(*) BYTE DATA (' FATAL ERROR. $$$'),
        I BYTE;
    I = INPUT(IN$STATUS) AND 07H;
    DO CASE I;
        ,
        CALL CRT$PRINT$STRING( MSG1);
        CALL CRT$PRINT$STRING( MSG2);
        CALL CRT$PRINT$STRING( MSG3);
        CALL CRT$PRINT$STRING( MSG4);
        CALL CRT$PRINT$STRING( MSG5);
    
```

FLOATING\$POINT

```
CALL CRT$PRINT$STRING(, MSG6);  
END;  
CALL CRT$PRINT$STRING(, ERROR);  
CALL SEND$CRLF;  
RETURN;  
END FLOAT$MSG$ERROR;
```

FLOAT\$MSG\$ERROR

FLOATING\$POINT

CHECK

```

/*****
*
*
* CHECK:
* PROCEDURE USED TO CHECK FOR STATUS OF F. P. BOARD AND TO DETECT IF
* ANY ERROR HAS OCCURRED.
*
* USAGE:
* UNTYPED PROCEDURE THAT IS CALLED BY ALL PROCEDURES THAT TRY TO EXECUTE
* A FLOATING POINT OPERATION WITH THE F. P. BOARD.
*
*****/
CHECK: PROCEDURE
  DCL I BYTE;
  DO WHILE ((I:=INPUT(IN$FLAG)) AND BUSY$MASK) = BUSY$MASK
  END;
  IF (I AND ERROR$MASK) = ERROR$MASK
  THEN DO;
    CALL FLOAT$MSG$ERROR;
    HALT;
  END;
  RETURN;
END CHECK;

```

FLOATING\$POINT

MUL

```

/*****
*
* MUL:
* PROCEDURE USED TO PERFORM FIXED POINT MULTIPLICATION USING THE F.P.
* BOARD.
*
* PARAMETERS:
* - A - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE FIRST
* OPERAND WILL BE LOCATED.
* - B - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE SECOND
* OPERAND WILL BE LOCATED.
* - C - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE RESULT
* IS DESIRED TO BE PLACED. NOTICE THAT IT COULD BE THE SAME
* LOCATION USED FOR ANY OF THE OPERANDS.
*
* *****/
MUL: PROCEDURE (A,B,C) PUBLIC;
  DCL (A,B,C) ADDRESS;
  CALL ADJUST$OP (A,B);
  OUTPUT<OUT$OP$CODE> = MUL$CODE;
  CALL CHECK;
  CALL VAL$RESULT (C);
  RETURN;
END MUL;

```

FLOATING\$POINT

DIV

```

/*****
*
* DIV:
* PROCEDURE USED TO PERFORM FIXED POINT DIVISION USING THE
* F.P. BOARD.
*
* PARAMETERS:
* - A - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE DIVI-
*   DEND IS LOCATED.
* - B - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE DIVI-
*   SOR IS LOCATED.
* - C - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE RESULT
*   (QUOTIENT) IS DESIRED TO BE PLACED.
* - R - POINTER TO A 2 BYTE VECTOR (ADDRESS VALUE) IN WHICH THE REMAIN-
*   DER IS DESIRED TO BE PLACED. NOTICE THAT C AND R COULD POINT TO ANY
*   OF THE TWO OPERANDS IF SO DESIRED.
*
*****/
DIV: PROCEDURE (A,B,C,R) PUBLIC;
  DCL (A,B,C,R) ADDRESS;
  CALL ADJUST$OP(A,B);
  OUTPUT(OUT$OP$CODE) = DIV$CODE;
  CALL CHECK;
  CALL VAL$RESULT#2(C,R);
  RETURN;
END DIV;

```

FLOATING\$POINT

EDIV

```

/*****
*
* EDIV:
* PROCEDURE USED TO PERFORM EXTENDED FIXED POINT DIVISION USING THE
* F. P. BOARD.
*
* PARAMETERS:
* - A - POINTER TO A 4 BYTE VECTOR THAT WILL PROVIDE THE DIVIDEND.
* - B - POINTER TO A 2 BYTE VECTOR THAT WILL PROVIDE THE DIVISOR.
* - C - POINTER TO A 4 BYTE VECTOR IN WHICH THE QUOTIENT WILL BE RETURNED.
* - R - POINTER TO A 4 BYTE VECTOR IN WHICH THE REMAINDER WILL BE RETURNED.
*
*****/
EDIV: PROCEDURE (A,B,C,R) PUBLIC;
DECL (A,B,C,R) ADDRESS,
      OP2 BASED B (2) BYTE,
      I BYTE;
CALL ADJUST1$OP (A);
DO I = 0 TO LAST(OP2);
  RES$TABLE(I + 4) = OP2(I);
END;
OUTPUT(OUT$OP$CODE) = EDIV$CODE;
CALL CHECK;
CALL VAL$RESULT$1 (C,R);
RETURN;
END EDIV;

```


FLOATING\$POINT

FMUL

```

/*****
*
* FMUL:
* PROCEDURE USED TO PERFORM FLOATING POINT MULTIPLICATION USING THE
* F. P. BOARD.
*
* PARAMETERS:
* -A,B,C. - POINTERS TO 3 FOUR BYTE VECTORS THAT POINT TO THE TWO OPERANDS
* AND THE RESULT RESPECTIVELY. NOTE THAT THE RESULT COULD BE THE SAME
* VECTOR USED TO INDICATE ANY OPERAND.
*
*****/
FMUL: PROCEDURE (A,B,C) PUBLIC;
    DCL (A,B,C) ADDRESS;
    CALL ADJUST$OP(A,B);
    OUTPUT(OUT$OP$CODE) = FMUL$CODE;
    CALL CHECK;
    CALL VAL$RESULT(C);
    RETURN;
END FMUL;

```

FLOATING\$POINT

FDIV

```

/*****
*
* FDIV:
*   PROCEDURE USED TO PERFORM FLOATING POINT DIVISION USING THE F.P BOARD.
*
* PARAMETERS:
*   - A,B,C - POINTERS TO 3 FOUR BYTE VECTORS THAT WILL POINT TO THE DIVIDEND
*             AND THE DIVISOR ON THE FIRST TWO, AND THE RESULT ON THE THIRD. NOTE
*             THAT THE RESULT COULD BE PUT IN THE SAME PLACE OF ANY OPERAND IF SO
*             DESIRED.
*
* *****/
FDIV: PROCEDURE (A,B,C) PUBLIC;
      DCL (A,B,C) ADDRESS;
      CALL ADJUST$OP (A,B);
      OUTPUT(OUT$OP$CODE) = FDIV$CODE;
      CALL CHECK;
      CALL VAL$RESULT(C);
      RETURN;
      END FDIV;

```

FLOATING\$POINT

FADD

```

/*****
*
* FADD:
*   PROCEDURE USED TO PERFORM FLOATING POINT ADDITION USING THE F.P. BOARD.
*
* PARAMETERS:
*   - A,B,C - POINTERS TO 3 FOUR BYTE VECTORS. THE FIRST TWO POINT TO THE
*     TWO OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR RESULT. NOTE
*     THAT THE RESULT COULD ALSO BE PLACED IN ANY OF THE OPERAND VECTORS.
*
*****/
FADD: PROCEDURE (A,B,C) PUBLIC;
      DCL (A,B,C) ADDRESS;
      CALL ADJUST$OP (A,B);
      OUTPUT(OUT$OP$CODE) = FADD$CODE;
      CALL CHECK;
      CALL VAL$RESULT (C);
      RETURN;
      END FADD;

```

FLOATING\$POINT

FSUB

```

/*****
*
* FSUB:
* PROCEDURE USED TO PERFORM FLOATING POINT SUBTRACTION USING THE
* F. P. BOARD.
*
* PARAMETERS:
* - A,B,C - POINTERS TO 3 FOUR BYTE VECTORS. THE FIRST TWO POINT TO
* BOTH OPERANDS, AND THE THIRD ONE POINTS TO THE VECTOR WHERE THE
* RESULT IS DESIRED TO BE PLACED. NOTE THAT THE RESULT COULD BE
* PLACED IN ANY OF BOTH OPERANDS.
*
*****/
FSUB: PROCEDURE (A,B,C) PUBLIC;
DECL (A,B,C) ADDRESS;
CALL ADJUST$OP (A,B);
OUTPUT(OUT$OP$CODE) = FSUB$CODE;
CALL CHECK;
CALL VAL$RESULT (C);
RETURN;
END FSUB;

```


FSQR

FLOATING\$POINT

```

/*****
*
* FSQR:
* PROCEDURE USED TO PERFORM A FLOATING POINT SQUARE USING THE F. P.
* BOARD.
*
* PARAMETERS:
* - A, C - POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VECTOR IN WHICH
* A FLOATING POINT QUANTITY IS LOCATED AND TO WHICH ITS SQUARE WILL BE
* OBTAINED. C POINTS TO A VECTOR IN WHICH THE RESULT IS DESIRED TO BE
* PLACED. NOTE THAT A AND C COULD POINT TO THE SAME VECTOR.
*
*****/
FSQR: PROCEDURE (A,C) PUBLIC;
  DCL (A,C) ADDRESS;
  CALL ADJUST1$OF(A);
  OUTPUT(OUT$OP$CODE) = FSQR$CODE;
  CALL CHECK;
  CALL VAL$RESULT (C);
  RETURN;
END FSQR;

```

FLOATING\$POINT

FLTDS

```

/*****
*
* FLTDS:
*   PROCEDURE USED TO PERFORM A FIXED-TO-FLOAT CONVERSION USING THE F. P.
*   BOARD.
*
* PARAMETERS:
*   - A, C - POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VECTOR CONTAINING
*     A FIXED POINT INTEGER AND C POINTS TO A VECTOR WHERE THE SINGLE PRE-
*     CISION FLOATING POINT REPRESENTATION OF THE SAME VALUE, IS DESIRED TO
*     BE PLACED. NOTE THAT BOTH, A AND C, COULD POINT TO THE SAME VECTOR.
*
*****/
FLTDS: PROCEDURE (A,C) PUBLIC;
  DCL (A,C) ADDRESS;
  CALL ADJUST1$OP (A);
  OUTPUT(OUT$OP$CODE) = FLTDS$CODE;
  CALL CHECK;
  CALL VAL$RESULT (C);
  RETURN;
END FLTDS;

```

FLOATING\$POINT

FIXSD

```

/*****
*
* FIXSD:
* PROCEDURE USED TO PERFORM A FLOAT-TO-FIXED CONVERSION USING THE F.P.
* BOARD.
*
* PARAMETERS:
* - A,C - POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO A VECTOR CONTAINING
* THE FIXED POINT QUANTITY DESIRED TO BE CONVERTED. C POINTS TO A VECTOR
* IN WHICH THE CONVERSION IS DESIRED TO BE PLACED. NOTE THAT BOTH COULD
* POINT TO THE SAME VECTOR.
*
* *****/
FIXSD: PROCEDURE (A,C) PUBLIC;
DECL (A,C) ADDRESS;
CALL ADJUST1$OP (A);
OUTPUT(OUT$OP$CODE) = FIXSD$CODE;
CALL CHECK;
CALL VAL$RESULT (C);
RETURN;
END FIXSD;

```

FLOATING\$POINT

FSQRT

```

/*****
*
* FSQRT:
* PROCEDURE TO PERFORM THE SQUARE ROOT OF A FLOATING POINT NUMBER USING
* THE F. P. BOARD.
*
* PARAMETERS:
* - A,C - POINTERS TO 2 FOUR BYTE VECTORS. A POINTS TO THE VECTOR CONTAINING
* THE NUMBER TO WHICH ITS SQUARE ROOT WILL BE OBTAINED. C POINTS TO THE
* VECTOR IN WHICH THE RESULT WILL BE PLACED. NOTE THAT BOTH POINTERS
* COULD BE REFERING TO THE SAME VECTOR.
*
* *****/
FSQRT: PROCEDURE (A,C) PUBLIC;
  DCL (A,C) ADDRESS;
  CALL ADJUST1$OP (A);
  OUTPUT(OUT$OP$CODE) = FSQRT$CODE;
  CALL CHECK;
  CALL VAL$RESULT (C);
  RETURN;
END FSQRT;

```


FLOATING\$POINT

FCMPR

```

/*****
*
* FCMPR:
* PROCEDURE USED TO COMPARE TWO FLOATING POINT NUMBERS USING THE F.P. BOARD.
*
* PARAMETERS:
* - A,B - POINTERS TO 2 FOUR BYTE VECTORS CONTAINING THE TWO FLOATING
*   POINT NUMBERS DESIRED TO BE COMPARED.
* - C - POINTS TO A BYTE VALUE CONTAINING THE CODE CORRESPONDING TO THE
*   TYPE OF COMPARISON DESIRED:
*       < ..... 0
*       <= ..... 1
*       > ..... 2
*       >= ..... 3
*       = ..... 4
*       <> ..... 5
*
* USAGE:
* TYPED PROCEDURE. IF THE RELATION DESIRED TO BE TESTED HOLDS, A VALUE OF
* '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FALSE) WILL BE
* RETURNED.
*
*****/
FCMPR: PROCEDURE (A,B,C) BYTE PUBLIC;
  DCL (A,B,C) ADDRESS;
  (TEST BASED C, RESULT, FLAG) BYTE;
  FLAG = 000H; /* RESET FLAG USED TO CHECK TWO NEGATIVE NUMBERS. */
  CALL ADJUST$OP (A,B);
  IF (RES$TABLE(3) >= 80H) AND
    (RES$TABLE(7) >= 80H)
  THEN FLAG = 0FFH;

```

FLOATING\$POINT

FCMPR

```
OUTPUT(OUT$OP$CODE) = FCMPR$CODE;  
CALL CHECK;  
RESULT = INPUT (IN$STATUS) AND 0E0H;  
IF FLAG AND (RESULT <> 80H)  
THEN DO;  
  IF RESULT = 40H  
  THEN RESULT = 20H;  
  ELSE RESULT = 40H;  
END;  
RETURN (RESULT := COMPARE(RESULT, TEST));  
END FCMPR;
```

FLOATING\$POINT

FZTST

```

/*****
*
* FZTST:
*   PROCEDURE USED TO TEST A FLOATING POINT NUMBER AGAINST 0.0 USING THE F.P.
*   BOARD.
*
* PARAMETERS:
*   - FI - POINTER TO A FOUR BYTE VECTOR CONTAINING THE FLOATING POINT VALUE
*     DESIRED TO BE TESTED.
*   - C - POINTER TO A BYTE VALUE CONTAINING THE CODE OF THE COMPARISON DESI-
*     RED, ACCORDING TO THE FOLLOWING RULES:
*
*       < ..... 0
*       <= ..... 1
*       / ..... 2
*       >= ..... 3
*       = ..... 4
*       <> ..... 5
*
* USAGE:
*   TYPED PROCEDURE. IF THE RELATION DESIRED TO BE TESTED HOLDS, A VALUE OF
*   '01H' (TRUE) IS RETURNED, OTHERWISE A VALUE OF '00H' (FALSE) WILL BE RE-
*   TURNED.
*
*****/
FZTST: PROCEDURE (A,C) BYTE PUBLIC;
  DCL (A,C) ADDRESS,
  OP1 BASED A (4) BYTE,
  (TEST BASED C, RESULT, I) BYTE;
  DCL LOWER$BOUND (4) BYTE DATA (0E2H,0FAH,0FFH,0B4H), /* -0.00000009 */
  UPPER$BOUND (4) BYTE DATA (0E2H,0FAH,0FFH,034H), /* 0.00000009 */
  LOW BYTE DATA (01H), /* LESS THAN OR EQUAL */

```

FLOATING\$POINT

FZTST

```

HIGH BYTE DATA (03H); /* GREATER THAN OR EQUAL */

/* CHECK IF NUMBER IS IN BOUNDARIES OF DEFINED SYSTEM ZERO */
IF (TEST <> 0) AND (TEST <> 2) AND
  (FCMPR( OPL, LOWER$BOUND, HIGH)) AND
  (FCMPR( OPL, UPPER$BOUND, LOW))
THEN DO;
  DO I = 0 TO 3;
    OP1(I) = 00H;
  END;
END;

CALL ADJUST1$OP (A);
OUTPUT(OUT$OP$CODE) = FZTST$CODE;
CALL CHECK;
RESULT = INPUT (IN$STATUS) AND 000H;
RETURN (RESULT:= COMPARE(RESULT,TEST));
END FZTST;

```


FLOATING\$POINT

EXCH

```

/*****
*
* EXCH:
* PROCEDURE USED TO EXCHANGE TWO FLOATING POINT VALUES USING THE F.P.
* BOARD.
*
* PARAMETERS:
* - A,C - POINTERS TO 2 FOUR BYTE VECTORS CONTAINING TWO FLOATING
* POINT NUMBERS DESIRED TO BE EXCHANGED.
*
*****/
EXCH: PROCEDURE (A,C) PUBLIC;
      DCL (A,C) ADDRESS;
      CALL ADJUST$OP (A,C);
      OUTPUT<OUT$OP$CODE> = EXCH$CODE;
      CALL CHECK;
      CALL VAL$RESULT$1(A,C);
      RETURN;
      END EXCH;

```

FLOATING\$POINT

COS\$SIN

```

/*****
*
* COS$SIN:
* THIS PROCEDURE IS USED TO CALCULATE THE COSINE AND SINE FUNCTIONS
* OF A GIVEN ARGUMENT IN RADIANS.
*
* PARAMETERS:
* - A - POINTER TO A FOUR BYTE VECTOR IN WHICH THE FLOATING POINT REPRESENTATION OF AN ANGLE IN RADIANS IS LOCATED.
*
* - C - POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE COSINE OF THE GIVEN ANGLE, WILL BE PLACED.
*
* - S - POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE SINE OF THE GIVEN ANGLE WILL BE PLACED.
*
*****/
COS$SIN: PROCEDURE (A,C,S) PUBLIC;
    DCL (A,C,S) ADDRESS,
        ANGLE BASED A (4) BYTE,
        COSINE BASED C (4) BYTE,
        SINE BASED S (4) BYTE,
        ANGLE$SQ (4) BYTE,
        TEMP (4) BYTE,
        TEMP0 (4) BYTE,
        TEMP1 (4) BYTE,
        MINUS$ONE (4) BYTE DATA (00H,00H,80H,00FH),
        ONE$FLOAT (4) BYTE DATA (00H,00H,80H,3FH),
        TWO$FLOAT (4) BYTE DATA (00H,00H,00H,40H),
        PI$FLOAT (4) BYTE DATA (008H,0FH,49H,40H),
        TWO$PI (4) BYTE DATA (008H,0FH,0C9H,40H),
        /* -1.0 */
        /* 1.0 IN F.P. FORMAT. */
        /* 2.0 */
        /* 3.141593 */
        /* 6.2831853 */

```

FLOATING\$POINT

COS\$SIN

```

PI$OVER2 (4) BYTE DATA (00BH,0FH,0C9H,3FH), /* 1.5707963 */
PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H), /* 4.7123889 */
CONST$1 (4) BYTE DATA (0B5H,1FH,12H,3FH), /* 0.5707963 */
CONST$2 (4) BYTE DATA (0E6H,5DH,25H,0BFH), /* -0.645964 */
CONST$3 (4) BYTE DATA (0E1H,35H,0A3H,3DH), /* 0.079629261 */
CONST$4 (4) BYTE DATA (0AAH,68H,99H,0BBH), /* -0.0046816668 */
CONST$5 (4) BYTE DATA (25H,0BH,28H,39H), /* 0.0001602588 */
CONST$6 (4) BYTE DATA (0A4H,067H,66H,0B6H), /* -0.0000034333 */
CHECK BYTE DATA (002H), /* CHECK FOR GREATER THAN */
CHECK1 BYTE DATA (003H), /* CHECK FOR GREATER THAN OR EQUAL */
CHECK2 BYTE DATA (001H), /* CHECK FOR LESS THAN OR EQUAL */
CHECK3 BYTE DATA (004H), /* CHECK FOR EQUAL */
(SIGN, SIGN1, QUAD, I) BYTE;

DO I = 0 TO 3;
  TEMP0(I) = ANGLE(I);
  SINE(I), COSINE(I) = 00H;
  END;
/* CHECK IF ANGLE IS >= 360 DEGREES. */
SIGN = FCMPR(, TEMP0, , TWO$PI, , CHECK1);
DO WHILE SIGN;
  CALL FSUB(, TEMP0, , TWO$PI, , TEMP0);
  SIGN = FCMPR(, TEMP0, , TWO$PI, , CHECK1);
  END;
/* CHECK IF ANGLE IS NEGATIVE. */
DO WHILE TEMP0(3) >= 080H;
  CALL FADD(, TEMP0, , TWO$PI, , TEMP0);
  END;
/* CHECK FOR SPECIAL CASES */
IF FCMPR(, TEMP0, , PI$OVER2, , CHECK3)
  THEN DO; /* 90 DEGREES */

```

FLOATING\$POINT

COS\$SIN

```

DO I = 0 TO 3;
  SINE(I) = ONE$FLOAT(I);
END;
RETURN;
END;
IF FCMPR (. TEMP0, . PI$3$OVER2, . CHECK3)
THEN DO; /* 270 DEGREES */
DO I = 0 TO 3;
  SINE(I) = MINUS$ONE(I);
END;
RETURN;
END;
IF FCMPR (. TEMP0, . TWO$PI, . CHECK3)
THEN DO; /* 360 DEGREES */
DO I = 0 TO 3;
  COSINE(I) = ONE$FLOAT(I);
END;
RETURN;
END;
/* TO NORMALIZE THE ANGLE BETWEEN 0 AND 90 DEGREES. */
QUAD = 1;
IF (SIGN := FCMPR(. TEMP0, . PI$4$OVER2, . CHECK)) AND
(SIGN1 := FCMPR(. TEMP0, . PI$F$FLOAT, . CHECK2))
THEN DO;
  QUAD = 2;
  CALL FSUB(. PI$F$FLOAT, . TEMP0, . TEMP0);
END;
ELSE DO;
  IF (SIGN := FCMPR(. TEMP0, . PI$F$FLOAT, . CHECK)) AND
  (SIGN1 := FCMPR(. TEMP0, . PI$3$OVER2, . CHECK2))
  THEN DO;

```


FLOFITING\$POINT

COS\$SIN

```

QUAD = 3;
CALL FSUB(.TEMP0,.PI$FLOAT,.TEMP0);
END;
ELSE DO;
  IF (SIGN := FCMPR(.TEMP0,.PI$3$OVER2,.CHECK))
    THEN DO;
      QUAD = 4;
      CALL FSUB(.TWO$PI,.TEMP0,.TEMP0);
    END;
  END;
END;

/* CONVERT ANGLE IN RADIANS TO ANGLE IN SEMICIRCLE UNITS. */
CALL FDIV(.TEMP0,.PI$FLOAT,.TEMP);
/* GET THE SQUARE OF THE ANGLE. */
CALL FSQR(.TEMP,.ANGLE$SQ);
/* PERFORM HASTINGS APPROXIMATION. */
CALL FMUL(.ANGLE$SQ,.CONST$6,.TEMP1);
CALL FADD(.TEMP1,.CONST$5,.TEMP1);
CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
CALL FADD(.TEMP1,.CONST$4,.TEMP1);
CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
CALL FADD(.TEMP1,.CONST$3,.TEMP1);
CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
CALL FADD(.TEMP1,.CONST$2,.TEMP1);
CALL FMUL(.ANGLE$SQ,.TEMP1,.TEMP1);
CALL FADD(.TEMP1,.CONST$1,.TEMP1);
CALL FMUL(.TEMP1,.TEMP1,.TEMP1);
CALL FADD(.TEMP1,.TEMP1,.TEMP1);
CALL FSQR(.TEMP1,.TEMP1);
/* TO COMPUTE THE VALUE OF THE COSINE. */

```

FLOATING\$POINT

COS\$SIN

```

CALL FSUB(.ONE$FLOAT, TEMP1, COSINE);
/* TO COMPUTE THE VALUE OF THE SINE. */
CALL FSQR(COSINE, TEMP1);
CALL FSUB(.ONE$FLOAT, TEMP1, TEMP1);
CALL FSQRT(TEMP1, SINE);
/* GIVE SIGNS TO COSINE AND SINE VALUES ACCORDING TO QUADRANT */
DO CASE QUAD;
;
;
COSINE(3) = COSINE(3) XOR 80H; /* FIRST QUADRANT */
DO; /* SECOND QUADRANT */
COSINE(3) = COSINE(3) XOR 80H; /* THIRD QUADRANT */
SINE(3) = SINE(3) XOR 80H;
END;
SINE(3) = SINE(3) XOR 80H; /* FOURTH QUADRANT */
END;
/* ALL DONE. RETURN. */
END COS$SIN;

```

FLOATING\$POINT

ARC\$TAN

```

/*****
*
* ARC$TAN:
* THIS PROCEDURE IS USED TO CALCULATE THE ARC$TAN FUNCTION IN RADIAN
* GIVEN AS ARGUMENT A RATIO OF TWO VALUES IN FP REPRESENTATION.
*
* PARAMETERS:
* - X - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE DENOMINATOR
* OF THE RATIO.
* - Y - POINTER TO A FOUR BYTE VECTOR REPRESENTING THE NUMERATOR
* OF THE RATIO.
* - A - POINTER TO A FOUR BYTE VECTOR IN WHICH THE VALUE OF THE ANGLE
* (IN RADIAN) WILL BE PLACED AFTER CALCULATION OF THE ARC$TAN.
*
*****
ARC$TAN: PROCEDURE (X,Y,A) PUBLIC;
DCL (X,Y,A) ADDRESS,
DELTA$X BASED X (4) BYTE,
DELTA$Y BASED Y (4) BYTE,
ANGLE BASED A (4) BYTE,
PI$FLOAT (4) BYTE DATA (0DBH,0FH,43H,40H),
TWO$PI (4) BYTE DATA (0DBH,0FH,0C9H,40H),
PI$OVER4 (4) BYTE DATA (0DBH,0FH,43H,3FH),
PI$OVER2 (4) BYTE DATA (0DBH,0FH,0C9H,3FH),
PI$3$OVER2 (4) BYTE DATA (0E4H,0CBH,96H,40H),
CONST$1 (4) BYTE DATA (0F5H,0FFH,7FH,3FH),
CONST$2 (4) BYTE DATA (1CH,0A6H,0AAH,0BEH),
CONST$3 (4) BYTE DATA (0A7H,40H,4CH,3EH),
CONST$4 (4) BYTE DATA (63H,6CH,0EH,0BEH),
CONST$5 (4) BYTE DATA (0DEH,77H,0C5H,3DH),
CONST$6 (4) BYTE DATA (0C4H,01H,65H,0EDH),
/* 3.141593 */
/* 6.2831853 */
/* 0.78539819 */
/* 1.5707963 */
/* 4.7123889 */
/* 0.99999993329 */
/* -0.3332985605 */
/* 0.1994653599 */
/* -0.1390853351 */
/* 0.0964200441 */
/* -0.0559098861 */

```

FLOATING\$POINT

ARC\$TAN

```

CONST$7 (4) BYTE DATA (51H, 16H, 0B3H, 3CH),
CONST$8 (4) BYTE DATA (0E6H, 0D7H, 84H, 0BBH),
CHECK BYTE DATA (04H),
MSG1(*) BYTE DATA
    ( ' ARC$TAN FUNCTION UNDEFINED FOR BOTH ARGUMENTS EQUAL TO ZERO. $$' ),
MSG2(*) BYTE DATA ( ' FATAL ERROR. $$' ),
Z (4) BYTE,
Z$SQUARE (4) BYTE,
TEMP (4) BYTE,
TEMP1 (4) BYTE,
(SIGN$X, SIGN$Y, ZERO$X, ZERO$Y, I) BYTE,
SIGN$X, SIGN$Y = 00H,
DO I = 0 TO 3,
    TEMP(I) = DELTA$Y(I),
    TEMP1(I) = DELTA$X(I),
END,
/* SAVE SIGN TO DETERMINE QUADRANT */
IF TEMP(3) >= 80H THEN SIGN$Y = 0FFH,
IF TEMP(3) >= 80H THEN SIGN$X = 0FFH,
/* CHECK FOR VALID ARGUMENTS */
IF (ZERO$Y := FZTST(.DELTA$Y, .CHECK)) AND
(ZERO$X := FZTST(.DELTA$X, .CHECK))
THEN DO,
    CALL CRT$PRINT$STRING (.MSG1),
    CALL CRT$PRINT$STRING (.MSG2),
    HALT,
END,
IF ZERO$X
THEN DO,
    IF SIGN$Y
    THEN DO,

```


FLOATING\$POINT

ARC\$TAN

```

DO I = 0 TO 3;
  ANGLE(I) = PI$3$OVER2(I);
END;
RETURN;
END;
ELSE DO;
  DO I = 0 TO 3;
    ANGLE(I) = PI$OVER2(I);
  END;
  RETURN;
END;
END;

/* FORM 2 TO PERFORM HASTINGS APPROXIMATION */
TEMP(3) = TEMP(3) AND 7FH;
TEMP1(3) = TEMP1(3) AND 7FH;
CALL FSUB(TEMP, .TEMP1, .Z);
CALL FADD(TEMP, .TEMP1, .TEMP);
CALL FDIV(TEMP, .TEMP, .Z);
CALL FSQR(TEMP, .Z$SQUARE);
/* PERFORM HASTINGS APPROXIMATION FOR ARC$TAN */
CALL FMUL(TEMP, .Z$SQUARE, .CONST$8, .TEMP);
CALL FADD(TEMP, .CONST$7, .TEMP);
CALL FMUL(TEMP, .Z$SQUARE, .TEMP, .TEMP);
CALL FADD(TEMP, .CONST$6, .TEMP);
CALL FMUL(TEMP, .Z$SQUARE, .TEMP, .TEMP);
CALL FADD(TEMP, .CONST$5, .TEMP);
CALL FMUL(TEMP, .Z$SQUARE, .TEMP, .TEMP);
CALL FADD(TEMP, .CONST$4, .TEMP);
CALL FMUL(TEMP, .Z$SQUARE, .TEMP, .TEMP);
CALL FADD(TEMP, .CONST$3, .TEMP);
CALL FMUL(TEMP, .Z$SQUARE, .TEMP, .TEMP);

```

ARC\$TAN

FLOATING\$POINT

```
CALL FADD(.TEMP, .CONST$2, .TEMP);
CALL FMUL(.Z$SQUARE, .TEMP, .TEMP);
CALL FADD(.TEMP, .CONST$1, .TEMP);
CALL FMUL(.Z, .TEMP, .TEMP);
CALL FADD(.TEMP, .PI$OVER4, .ANGLE);
/* RESTORE ANGLE TO PROPER QUADRANT */
IF (NOT SIGN$Y) AND SIGN$X /* SECOND QUADRANT */
    THEN CALL FSUB(.PI$FLOAT, .ANGLE, .ANGLE);
IF SIGN$Y AND SIGN$X /* THIRD QUADRANT */
    THEN CALL FADD(.PI$FLOAT, .ANGLE, .ANGLE);
IF SIGN$Y AND (NOT SIGN$X) /* FOURTH QUADRANT */
    THEN CALL FSUB(.TWO$PI, .ANGLE, .ANGLE);
/* ALL DONE. RETURN */
END ARC$TAN;
```

END FLOATING\$POINT;

TIME

TIME

TIME: DO;

CRT\$WRITE:
PROCEDURE (A) EXTERNAL;
DECLARE A BYTE; END;

CRT\$PRINT\$STRING:
PROCEDURE (A) EXTERNAL;
DECLARE A ADDRESS; END;

ECHO\$CRT:
PROCEDURE BYTE EXTERNAL;
END;

SEND\$BEL:
PROCEDURE EXTERNAL;
END;

SEND\$CR:
PROCEDURE EXTERNAL;
END;

SEND\$CRLF:
PROCEDURE EXTERNAL;
END;

SEND\$SUB:
PROCEDURE EXTERNAL;
END;

TIME

TIME

```
GET$BYTE:
  PROCEDURE (A) BYTE EXTERNAL;
  DECLARE A BYTE; END;
```

```
CHECK$INPUT:
  PROCEDURE BYTE EXTERNAL;
  END;
```

```
CLEAR$LOW$SCREEN:
  PROCEDURE EXTERNAL;
  END;
```

```
DECLARE (MILI$SEC,DUMMY$SEC,SECONDS,MINUTES,HOURS,DAY,SEC$TIME) BYTE PUBLIC,
        TIME$STEP ADDRESS PUBLIC,
        TIME$BUFFER(6) BYTE PUBLIC;
```


TIME

CLOCK

```

/*****
*
* CLOCK:
* THIS PROCEDURE IS OF THE TYPE INTERRUPT. IT IS USED TO MAINTAIN A REAL
* TIME HOUR, ONCE INITIATED. IT USES THE MDS REAL TIME CLOCK.
*
* USAGE:
* THIS PROCEDURE IS CALLED EACH TIME AN INTERRUPT FROM THE REAL TIME CLOCK
* IN THE MDS SYSTEM, IS PRODUCED.
*
* *** WARNING ***
* SINCE THE DEVELOPMENT OF THIS PROCEDURE WAS DONE UNDER ISIS, THE PROCEDURE
* HAD TO BE DECLARED AS INTERRUPT 7, SINCE ISIS DOES NOT ALLOW FOR INTERRUPTS
* LESS THAN OR EQUAL TO 2. THE INTERRUPT FROM THE REAL TIME CLOCK IS OF LEVEL
* 1, AND THEREFORE, A 'CALL MOVE' INSTRUCTION HAD TO BE IMPLEMENTED, IN
* ORDER TO MOVE THE CODE IN INT 7 TO INT 1 AND PASS OVER THIS ISIS INCONVENIEN-
* CE. IF A COPY OF THIS PROGRAM IS TO BE EXECUTED WITHOUT ISIS, THEN THIS PRO-
* CEDURE MUST BE RECOMPILED AS PROCEDURE INTERRUPT 1, AND THE 'CALL MOVE' STA-
* TEMENT IN THE INITIATE$CLOCK PROCEDURE, REMOVED.
*
*****/
CLOCK: PROCEDURE INTERRUPT 7;
  DECLARE TEMP BYTE;
  /* TO RESET THE MDS REAL TIME CLOCK. */
  OUTPUT(0FFH) = 03H;
  MILI$SEC = MILI$SEC + 1;
  IF MILI$SEC = 128 THEN DO;
    MILI$SEC = 0;
    DUMMY$SEC = DUMMY$SEC + 1;
  END;
  IF DUMMY$SEC = 08H THEN DO;

```

TIME

CLOCK

```

MILI$SEC, DUMMY$SEC = 0;
SEC$TIME = 0FFH; /* BOOLEAN VARIABLE. A SECOND HAS ELAPSED. */
SECONDS = SECONDS + 1;
TIME$STEP = TIME$STEP + 1;
IF SECONDS = 60 THEN DO;
  SECONDS = 00;
  MINUTES = MINUTES + 1;
  IF MINUTES = 60 THEN DO;
    MINUTES = 00;
    HOURS = HOURS + 1;
    IF HOURS = 24 THEN DO;
      HOURS = 00;
      DAY = 1;
    END;
  END;
END;
END;

/* DISABLE INTERRUPTS. */
DISABLE;
/* RESTORE CURRENT OPERATING LEVEL. */
OUTPUT(0FDH) = 020H;
/* SET THE MDS REAL TIME CLOCK. */
TEMP = INPUT(0FFH);
TEMP = INPUT(0FFH);
OUTPUT(0FFH) = 00H;
/* THE RETURN STATEMENT WILL ENABLE INTERRUPTS AUTOMATICALLY. */
RETURN;
END CLOCK;

```

TIME

INITIATE\$TIME

```

/*****
*
* INITIATE$TIME:
* PROCEDURE USED DURING SYSTEM INITIALIZATION. USED TO SET THE SIMULATED
* REAL TIME CLOCK TO AN SPECIFIED HOUR.
*
* USAGE:
* UTILIZES THE GLOBAL VARIABLES HOURS, MINUTES AND SECONDS.
*
*****/
INITIATE$TIME: PROCEDURE PUBLIC;
DECLARE OK BYTE;
DECLARE MSG(*) BYTE DATA (' *** BAD FORMAT. ***$'$);
OK = 0;
DO WHILE OK = 0;
HOURS, MINUTES, SECONDS = 0FFH;
CALL CRT$PRINT$STRING('INPUT THE TIME AS REQUESTED. $$');
CALL SEND$CRLF;
DO WHILE HOURS >= 24;
CALL CRT$PRINT$STRING('HOURS: $$');
HOURS = GET$BYTE(2);
IF HOURS >= 24
THEN DO;
CALL CRT$PRINT$STRING('MSG');
CALL SEND$BEL;
CALL SEND$CR;
CALL SEND$SUB;
END;
ELSE DO;
CALL CRT$WRITE(17H);
CALL SEND$CRLF;
/* ERASE TO END OF LINE */

```

TIME

INITIATE\$TIME

```
END;
DO WHILE MINUTES >= 60;
CALL CRT$PRINT$STRING(, ('MINUTES: $$'));
MINUTES = GET$BYTE(2);
IF MINUTES >= 60
THEN DO;
CALL CRT$PRINT$STRING(, MSG);
CALL SEND$BEL;
CALL SEND$CR;
CALL SEND$SUB;
END;
ELSE DO;
CALL CRT$WRITE(17H); /* ERASE TO END OF LINE */
CALL SEND$CRLF;
END;
END;
DO WHILE SECONDS >= 60;
CALL CRT$PRINT$STRING(, ('SECONDS: $$'));
SECONDS = GET$BYTE(2);
IF SECONDS >= 60
THEN DO;
CALL CRT$PRINT$STRING(, MSG);
CALL SEND$BEL;
CALL SEND$CR;
CALL SEND$SUB;
END;
ELSE DO;
CALL CRT$WRITE(17H); /* ERASE TO END OF LINE */
CALL SEND$CRLF;
END;
END;
```


INITIATE\$TIME

TIME

END;
OK = CHECK\$INPUT;
CALL CLEAR\$LOW\$SCREEN;
END;

END INITIATE\$TIME;

TIME

INITIATE\$CLOCK

```

/*****
*
* INITIATE$CLOCK:
* PROCEDURE USED TO START THE SIMULATED REAL TIME CLOCK.
*
*****/
INITIATE$CLOCK: PROCEDURE PUBLIC;
MILI$SEC, DUMMY$SEC = 00;

/*
*      *** WARNING ***
*
* THE FOLLOWING STATEMENT SHOULD BE REMOVED IF THIS MODULE IS TO BE
* EXECUTED WITHOUT ISIS. THE CLOCK PROCEDURE SHOULD ALSO BE RECOMPI-
* LED AS PROCEDURE INTERRUPT 1.
*
*/

CALL MOVE (3, 038H, 008H);
OUTPUT(0FDH) = 12H;
OUTPUT(0FCH) = 00H;
ENABLE;
OUTPUT(0FFH) = 00H;
END INITIATE$CLOCK;

```

TIME

ACTUAL\$TIME

```

/*****
*
* ACTUAL$TIME:
* PROCEDURE USED TO PUT INTO THE VECTOR 'TIME$BUFFER', THE ACTUAL
* TIME IN A STRING FORM.
*
*****/
ACTUAL$TIME: PROCEDURE PUBLIC;
    TIME$BUFFER(0) = HOURS/10 + 30H;
    TIME$BUFFER(1) = HOURS MOD 10 + 30H;
    TIME$BUFFER(2) = MINUTES/10 + 30H;
    TIME$BUFFER(3) = MINUTES MOD 10 + 30H;
    TIME$BUFFER(4) = SECONDS/10 + 30H;
    TIME$BUFFER(5) = SECONDS MOD 10 + 30H;
    END ACTUAL$TIME;

```

END TIME;

PLASMA\$PRIMITIVES

PLASMA\$PRIMITIVES

PLASMA\$PRIMITIVES: DO;

DECLARE LIT LITERALLY 'LITERALLY',
DCL LIT 'DECLARE';

DCL TRUE LIT '0FFH',
FALSE LIT '00H';

DCL PLASMA\$DATA LIT '04H',
PLASMA\$STATUS LIT '05H',
RECEIVE\$MASK LIT '06H',
TRANSMIT\$MASK LIT '05H';

DCL STATUS\$A LIT '04H',
RESET\$ALL LIT '00H',
OUT\$BUSY LIT '01H',
IN\$BUSY LIT '02H';

DCL STX LIT '02H',
ETX LIT '03H',
CS LIT '0CH',
CG LIT '0EH',
CV LIT '0FH',
EOL LIT '24H';

/* START TEXT */
/* ENABLE TEXT */
/* CLEAR SCREEN */
/* CONSTRUCT GRAPH */
/* CLEAR VECTORS */
/* END OF LINE */

PLASMA\$PRIMITIVES

DCL SET\$ERASE
SET\$DASHED
SET\$END

LIT '0100\$00000B',
LIT '0001\$00000B',
LIT '0010\$00000B';

PLASMA\$PRIMITIVES

PLASMA\$PRIMITIVES

SET\$STATUS\$PLASMA

```

/*****
*
* SET$STATUS$PLASMA:
* THIS PROCEDURE IS USED TO SET THE STATUS LINE FOR THE PLASMA.
* NOTE THAT THE LOGIC TO BE USED IS NEGATIVE.
*
* PARAMETERS:
* - STATUS. - ASCII CHARACTER USED TO DEFINE THE STATUS LINE.
*
*****/
SET$STATUS$PLASMA: PROCEDURE (STATUS) PUBLIC;
    DCL STATUS BYTE;
    OUTPUT(PLASMA$STATUS) = NOT STATUS;
    END SET$STATUS$PLASMA;

```

PLASMA\$PRIMITIVES

PLASMA\$WRITE

```

/*****
*
* PLASMA$WRITE:
* THIS PROCEDURE IS USED TO SEND A CHARACTER TO THE PLASMA DISPLAY.
*
* PARAMETERS:
* - CHAR. - ASCII CHARACTER DESIRED TO BE SENT.
*
*****/
PLASMA$WRITE: PROCEDURE (CHAR) PUBLIC;
DCL CHAR BYTE;
DO WHILE ((NOT INPUT(PLASMA$STATUS)) AND STATUS#A) <> STATUS#A;
END;
/* WAIT FOR PLASMA TO BE READY */
CALL SET$STATUS$PLASMA(RESET$ALL);
OUTPUT(PLASMA$DATA) = NOT CHAR;
CALL SET$STATUS$PLASMA(OUT$BUSY);
END PLASMA$WRITE;

```

PLASMA\$PRIMITIVES

CLEAR\$PLASMA

```

/*****
*
* CLEAR$PLASMA:
*   THIS PROCEDURE IS USED TO CLEAR THE PLASMA DISPLAY.
*
*****/
CLEAR$PLASMA: PROCEDURE PUBLIC;
  CALL SET$STATUS$PLASMA(IN$BUSY);
  CALL SET$STATUS$PLASMA(RESET$ALL);
  CALL PLASMA$WRITE(CS);
  CALL PLASMA$WRITE(CV);
  CALL SET$STATUS$PLASMA(RESET$ALL);
  END CLEAR$PLASMA;

```


PLASMA\$PRIMITIVES PLASMA\$WRITE\$VECTOR

```

/*****
*
* PLASMA$WRITE$VECTOR:
*   THIS PROCEDURE IS USED TO SEND A FOUR BYTE VECTOR TO THE PLASMA.
*
* PARAMETERS:
*   - A - POINTER TO THE FOUR BYTE VECTOR DESIRED TO BE SENT.
*
*****/
PLASMA$WRITE$VECTOR: PROCEDURE (A) PUBLIC;
  DCL A ADDRESS,
       VECTOR BASED A (4) BYTE,
       VPTR BYTE;
  DO VPTR = 0 TO 3;
    CALL PLASMA$WRITE(VECTOR(VPTR));
  END;
END PLASMA$WRITE$VECTOR;

```

PLASMA\$PRIMITIVES

PLASMA\$PRINT\$STRING

```

/*****
*
* PLASMA$PRINT$STRING:
* THIS PROCEDURE IS USED TO WRITE A GIVEN STRING IN A GIVEN POSITION AT
* THE PLASMA DISPLAY.
*
* PARAMETERS:
* - COLUMN - DENOTES THE COLUMN NUMBER DESIRED TO BE ADDRESSED.
* - ROW - DENOTES THE ROW NUMBER DESIRED TO BE ADDRESSED.
* - POINTER - POINTS TO THE FIRST BYTE OF THE STRING DESIRED TO BE DIS-
* PLAYED. NOTE THAT TWO CONSECUTIVE '$' SIGNS MUST MARK THE END OF
* THE STRING.
*
*****/
PLASMA$PRINT$STRING: PROCEDURE (COLUMN, ROW, POINTER) PUBLIC;
    DCL POINTER ADDRESS,
        BUFFER BASED POINTER (1) BYTE,
        (COLUMN, ROW, COUNT) BYTE;
    COUNT = 0;
    CALL SET$STATUS$PLASMA(IN$BUSY);
    CALL SET$STATUS$PLASMA(RESET$ALL);
    CALL PLASMA$WRITE(STX);
    CALL PLASMA$WRITE(COLUMN);
    CALL PLASMA$WRITE(ROW);
    DO WHILE (BUFFER(COUNT) <> EOL ) OR (BUFFER(COUNT + 1) <> EOL );
        CALL PLASMA$WRITE(BUFFER(COUNT));
        COUNT = COUNT + 1;
    END;
    CALL PLASMA$WRITE(ETX);
    CALL SET$STATUS$PLASMA(RESET$ALL);
    END PLASMA$PRINT$STRING;

```

PLASMA\$PRIMITIVES

INITIALIZE\$PLASMA

```

/*****
*
* INITIALIZE$PLASMA:
* THIS PROCEDURE IS USED TO INITIALIZE THE PLASMA DISPLAY.
*
*****/
INITIALIZE$PLASMA: PROCEDURE PUBLIC;
DCL BUFFER (*) BYTE DATA ('ON LINE.$$');
CALL CLEAR$PLASMA;
CALL PLASMA$PRINT$STRING(0, 2, .BUFFER);
END INITIALIZE$PLASMA;

```

PLASMA\$PRIMITIVES

SET\$VECTOR

```

/*****
*
* SET$VECTOR:
* THIS PROCEDURE IS USED PREPARE TWO X AND Y VALUES GIVEN INTO THE FOR-
* MAT REQUIRED BY THE PLASMA UNIT TO DEFINE A VECTOR.
* PARAMETERS:
* - X - ADDRESS VALUE.
* - Y - ADDRESS VALUE.
* - POINTER - POINTS TO A FOUR BYTE VECTOR IN WHICH THE X AND Y VALUES
* WILL BE ARRANGED.
*
*****
SET$VECTOR: PROCEDURE (X, Y, POINTER) PUBLIC;
DCL (X, Y, POINTER) ADDRESS,
    VECTOR BASED POINTER (4) BYTE;
VECTOR(0) = CG;
VECTOR(1), VECTOR(2), VECTOR(3) = 00H;
VECTOR(1) = LOW(X) AND 07FH;
VECTOR(2) = LOW(Y) AND 07FH;
VECTOR(3) = HIGH(SHL(Y AND 180H, 3)) OR
    HIGH(SHL(X AND 180H, 1));
VECTOR(3) = VECTOR(3) AND NOT SET$ERASE;
END SET$VECTOR;

```


PLASMA\$PRIMITIVES

START\$VECTOR\$SOLID

```

/*****
*
* START$VECTOR$SOLID:
* THIS PROCEDURE IS USED TO DEFINE A START POINT FOR A SOLID VECTOR.
*
* PARAMETERS:
* - X. - ADDRESS VALUE.
* - Y. - ADDRESS VALUE.
*
*****/
START$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
    DCL (X, Y) ADDRESS,
        VECTOR (4) BYTE;
    CALL SET$VECTOR(X, Y, VECTOR);
    VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
    VECTOR(3) = VECTOR(3) AND NOT SET$END;
    CALL PLASMA$WRITE$VECTOR(VECTOR);
    END START$VECTOR$SOLID;

```

PLASMA\$PRIMITIVES

STOP\$VECTOR\$SOLID

```

/*****
*
* STOP$VECTOR$SOLID:
* THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR A SOLID VECTOR.
*
* PARAMETERS:
* - X - ADDRESS VALUE.
* - Y - ADDRESS VALUE.
*
* *****/
STOP$VECTOR$SOLID: PROCEDURE (X, Y) PUBLIC;
DCL (X, Y) ADDRESS,
    VECTOR (4) BYTE;
CALL SET$VECTOR(X, Y, VECTOR);
VECTOR(3) = VECTOR(3) AND NOT SET$DASHED;
VECTOR(3) = VECTOR(3) OR SET$END;
CALL PLASMA$WRITE$VECTOR(. VECTOR);
END STOP$VECTOR$SOLID;
/*****/

```

PLASMA\$PRIMITIVES

START\$VECTOR\$DASH

```

/*****
*
* START$VECTOR$DASH:
* THIS PROCEDURE IS USED TO DEFINE A START POINT FOR A DASHED VECTOR.
*
* PARAMETERS:
* - X - ADDRESS VALUE.
* - Y - ADDRESS VALUE.
*
*****/
START$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
  DCL (X, Y) ADDRESS,
       VECTOR (4) BYTE;
  CALL SET$VECTOR(X, Y, VECTOR);
  VECTOR(3) = VECTOR(3) OR SET$DASHED;
  VECTOR(3) = VECTOR(3) AND NOT SET$END;
  CALL PLASMA$WRITE$VECTOR(VECTOR);
  END START$VECTOR$DASH;
*****/
```

PLASMA\$PRIMITIVES

STOP\$VECTOR\$DASH

```

/*****
*
* STOP$VECTOR$DASH:
*   THIS PROCEDURE IS USED TO DEFINE A STOP POINT FOR A DASHED VECTOR.
*
* PARAMETERS:
*   - X - ADDRESS VALUE.
*   - Y - ADDRESS VALUE.
*
*****/
STOP$VECTOR$DASH: PROCEDURE (X, Y) PUBLIC;
  DCL (X, Y) ADDRESS,
       VECTOR (4) BYTE;
  CALL SET$VECTOR(X, Y, VECTOR);
  VECTOR(3) = VECTOR(3) OR SET$DASHED;
  VECTOR(3) = VECTOR(3) OR SET$END;
  CALL PLASMA$WRITE$VECTOR(VECTOR);
  END STOP$VECTOR$DASH;
/
```


PLASMA\$PRIMITIVES

GRAPHIC\$DESIG

```

/*****
*
* GRAPHIC$DESIG:
* THIS PROCEDURE IS USED TO DISPLAY THE DESIGNATION OF A CONTACT IN THE
* NEAREST POSSIBLE ALPHANUMERIC LOCATION TO THE X, Y VALUES GIVEN.
*
* PARAMETERS:
* - X - ADDRESS VALUE.
* - Y - ADDRESS VALUE.
* - DESIG - POINTER TO THE DESIG OF A CONTACT.
*
*****
GRAPHIC$DESIG: PROCEDURE (X, Y, DESIG) PUBLIC;
DCL (X, Y, DESIG) ADDRESS,
    VALUE BASED DESIG ADDRESS,
    BUFFER (6) BYTE,
    (ROW, COLUMN) BYTE;
BUFFER(0) = ' ';
BUFFER(1) = VALUE / 100;
BUFFER(2) = VALUE MOD 100;
BUFFER(3) = ' ';
BUFFER(4), BUFFER(5) = EOL;
ROW = Y / 16;
COLUMN = ((X - 14) / 6) - 5;
IF X <= 8 THEN COLUMN = 0;
IF (X >= 9) AND (X <= 44) THEN COLUMN = X / 6;
IF (X >= 494) THEN COLUMN = 75;
CALL PLASMA$PRINT$STRING(COLUMN, ROW, BUFFER);
END GRAPHIC$DESIG;
/

```

AD-A059 603

NAVAL POSTGRADUATE SCHOOL MONTEREY CALIF

F/G 17/7

A MICROCOMPUTER BASED SHIPBOARD SURFACE-SUBSURFACE CONTACT PLOT--ETC(U)

JUN 78 A L GONCALVES, J E CUBA BRAVO

UNCLASSIFIED

NL

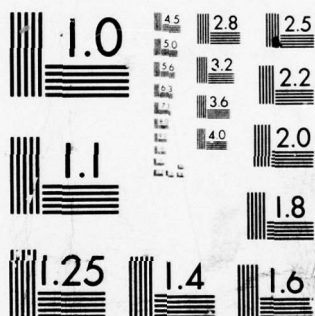
6 OF 6

AD
A059603



END
DATE
FILMED
12-78

DDC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

PLASMA\$PRIMITIVES

GRAPHIC\$DESIG

END PLASMA\$PRIMITIVES;

BASICS

BASICS

```

BASICS: DO;

DECLARE LIT LITERALLY 'LITERALLY',
        DCL LIT 'DECLARE';
DCL CR LIT '0DH',
    LF LIT '0AH',
    EOL LIT '24H',
    BS LIT '08H',
    SPACE LIT '20H',
    RUB LIT '7FH',
    SUB LIT '1AH',
    BEL LIT '07H';

DCL TRUE LIT '0FFH',
    FALSE LIT '00H',
    FOREVER LIT 'WHILE TRUE';

DCL CRTDATA LIT '0F6H',
    CRTSTATUS LIT '0F7H',
    CRT$REC$MASK LIT '0GH',
    CRT$TRT$MASK LIT '05H';

/* CARRIAGE RETURN. */
/* LINE FEED. */
/* $$ INDICATES END OF LINE. */
/* BACKSPACE. */
/* SPACE */
/* RUB OUT. */
/* UP ROW CURSOR. */
/* RING THE BELL. */

/* CRT DATA ON PORT 246. */
/* CRT STATUS ON PORT 247. */
/* MASK FOR CRT: RECEIVE. */
/* MASK FOR CRT: TRANSMIT. */

```

BASICS

CRT\$WRITE

```

/*****
*
* CRT$WRITE:
* PROCEDURE USED TO SEND A CHARACTER TO THE CRT.
*
* PARAMETERS:
* - CHAR. - CHARACTER BYTE TO BE SENT.
*
*****
CRT$WRITE: PROCEDURE (CHAR) PUBLIC;
DECL CHAR BYTE;
DO WHILE (INPUT(CRT$STATUS) AND CRT$TRT$MASK) <> CRT$TRT$MASK /WAIT*/
END;
OUTPUT(CRT$DATA) = CHAR;
END CRT$WRITE;

```

BASICS

CRT\$PRINT\$STRING

```

/*****
* CRT$PRINT$STRING:
* PROCEDURE USED TO SEND A STRING OF CHARACTERS TO THE CRT.
*
* PARAMETERS:
* - A - POINTER TO STRING. $$ WILL INDICATE END OF STRING. MAXIMUM
*   LENGTH: 80 CHARACTERS.
*
*****/
CRT$PRINT$STRING: PROCEDURE (A) PUBLIC;
  DCL (POINTER,A) ADDRESS,
    (BUFFER BASED A) (80) BYTE;
  POINTER = 0;
  DO WHILE (BUFFER(POINTER) <> EOL) OR (BUFFER(POINTER + 1) <> EOL);
    CALL CRT$WRITE(BUFFER(POINTER));
    POINTER = POINTER + 1;
  END;
  END CRT$PRINT$STRING;

```

BASICS

CRT\$READ

```

/*****
*
* CRT$READ:
* PROCEDURE USED TO RECEIVE A CHARACTER FROM THE CRT, AND RETURN ITS
* VALUE TO THE CALLING MODULE.
*
* USAGE:
* UNTYPED PROCEDURE. RETURNS A BYTE VALUE. (ASCII CODE)
*
*****/
CRT$READ: PROCEDURE BYTE PUBLIC;
DECL CHAR BYTE;
DO WHILE (INPUT(CRT$STATUS) AND CRT$REC$MASK) <> CRT$REC$MASK /*WAIT*/
END;
CHAR = INPUT(CRT$DATA);
IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
RETURN CHAR;
END CRT$READ;

```


BASICS

CRT\$TRY\$READ

```

/*****
*
* CRT$TRY$READ:
*   PROCEDURE USED TO TEST IF CHARACTER HAS BEEN SENT FROM CRT.
*
*   USAGE:
*     TYPED PROCEDURE. IF READING WAS SUCCESSFUL, ( A KEY WAS DEPRESSED )
*     THEN THE BYTE VALUE OF THE ASCII CHARACTER WILL BE RETURNED. NOTICE
*     THAT NO CHARACTER WILL BE DISPLAYED.
*     IF READING WAS NOT SUCCESSFUL, THEN A NUL ASCII CHARACTER (00H) IS
*     RETURNED, AND NO CHARACTER IS DISPLAYED.
*
*****/
CRT$TRY$READ: PROCEDURE BYTE PUBLIC;
  DCL CHAR BYTE;
  CHAR = 00H;
  IF (INPUT(CRT$STATUS) AND CRT$REC$MASK) = CRT$REC$MASK
    THEN CHAR = INPUT(CRT$DATA);
  IF CHAR >= 80H THEN CHAR = CHAR XOR 80H;
  RETURN CHAR;
END CRT$TRY$READ;

```

BASICS

ECHO\$CRT

```

/*****
*
* ECHO$CRT:
* PROCEDURE USED TO RECEIVE AN ASCII CHARACTER FROM THE CRT, DISPLAY
* THE SAME, AND RETURN ITS BYTE VALUE TO THE CALLING MODULE.
*
* USAGE:
* UNTYPED PROCEDURE. RETURNS A BYTE VALUE. (ASCII CODE.)
*
*****/
ECHO$CRT: PROCEDURE BYTE PUBLIC;
    DCL CHAR BYTE;
    CHAR = CRT$READ;
    CALL CRT$WRITE(CHAR);
    RETURN CHAR;
END ECHO$CRT;

```

SEND\$SUB

BASICS

```

/*****
*
* SEND$SUB:
* PROCEDURE USED TO SEND AN UP ROW CURSOR CHARACTER TO THE CRT.
*
*****/
SEND$SUB: PROCEDURE PUBLIC;
CALL CRT$WRITE(SUB);
END SEND$SUB;
```

SEND\$CR

BASICS

```
*****  
* SEND$CR:  
* PROCEDURE USED TO SEND A CR ASCII CHARACTER TO THE CRT.  
*  
*****  
SEND$CR: PROCEDURE PUBLIC;  
  CALL CRT$WRITE(CR);  
  END SEND$CR;
```


BASICS

SEND\$LF

```
/******  
* SEND$LF:  
* PROCEDURE USED TO SEND A LF ASCII CHARACTER TO THE CRT.  
*  
*****/  
SEND$LF: PROCEDURE PUBLIC;  
  CALL CRT$WRITE(LF);  
  END SEND$LF;
```

BASICS

SEND\$CRLF

```
/******  
* SEND$CRLF:  
* PROCEDURE USED TO SEND BOTH CR AND LF ASCII CHARACTERS TO CRT.  
*  
*****/  
SEND$CRLF: PROCEDURE PUBLIC;  
  CALL CRT$WRITE(CR);  
  CALL CRT$WRITE(LF);  
  END SEND$CRLF;
```

BASICS

SEND\$BEL

```
/******  
* SEND$BEL:  
* PROCEDURE USED TO SEND A 'BELL' ASCII CHARACTER TO THE CRT.  
*  
*****  
SEND$BEL: PROCEDURE PUBLIC;  
  CALL CRT$WRITE(BEL);  
  END SEND$BEL;
```

BASICS

SEND\$BS

```
/******  
*  
* SEND$BS:  
* PROCEDURE USED TO SEND A NON-DESTRUCTIVE BACKSPACE CHARACTER TO  
* THE CRT.  
*  
*****/  
SEND$BS: PROCEDURE PUBLIC;  
  CALL CRT$WRITE(BS);  
  END SEND$BS;
```


BASICS

SEND\$SPACE

```
/******  
* SEND$SPACE:  
* THIS PROCEDURE IS USED TO SEND SPACES TO THE CRT.  
*  
* PARAMETERS:  
* - NUM - NUMBER OF SPACES DESIRED.  
*  
*****  
SEND$SPACE: PROCEDURE (NUM) PUBLIC;  
  DCL NUM BYTE;  
  DO WHILE NUM > 0;  
    CALL CRT$WRITE(SPACE);  
    NUM = NUM - 1;  
  END;  
END SEND$SPACE;
```

BASICS

BYTE\$CHAR

```

/*****
*
* BYTE$CHAR:
* PROCEDURE USED TO DISPLAY THE DECIMAL VALUE OF A BYTE VARIA-
* BLE.
*
* PARAMETERS:
* - CHAR - BYTE VALUE DESIRED TO BE DISPLAYED.
*
*****/
BYTE$CHAR: PROCEDURE (CHAR) PUBLIC;
DCL VALUE(3) BYTE DATA (100,10,1);
DCL (I,CHAR,COUNT,TEMP) BYTE;
DO I = 0 TO LAST(VALUE);
    COUNT = 0;
    TEMP = VALUE(I);
    DO WHILE CHAR >= TEMP;
        CHAR = CHAR - TEMP;
        COUNT = COUNT + 1;
    END;
    CALL CRT$WRITE(COUNT + '0');
END;
END BYTE$CHAR;

```

BASICS

ADDRESS\$CHAR

```

/*****
*
* ADDRESS$CHAR:
* PROCEDURE USED TO DISPLAY THE DECIMAL VALUE OF AN ADDRESS
* VARIABLE.
*
* PARAMETERS:
* - CHAR - ADDRESS VALUE DESIRED TO BE DISPLAYED.
*
*****/
ADDRESS$CHAR: PROCEDURE (CHAR) PUBLIC;
DCL VALUE(5) ADDRESS DATA (10000,1000,100,10,1);
DCL (1,COUNT) BYTE,
    (CHAR,TEMP) ADDRESS;
DO I = 0 TO LAST(VALUE);
    COUNT = 0;
    TEMP = VALUE(I);
    DO WHILE CHAR >= TEMP;
        CHAR = CHAR - TEMP;
        COUNT = COUNT + 1;
    END;
    CALL CRT$WRITE(COUNT + '0');
END;
END ADDRESS$CHAR;

```

BASICS

BYTE\$TO\$ASCII

```

/*****
*
* BYTE$TO$ASCII:
* THIS PROCEDURE IS USED TO CONVERT A BYTE QUANTITY INTO TWO ASCII
* CHARACTERS REPRESENTING ITS HEXADECIMAL VALUE.
*
* PARAMETERS:
* - A - POINTER TO THE BYTE QUANTITY DESIRED TO BE CONVERTED.
* - B,C - POINTERS TO TWO BYTE QUANTITIES IN WHICH THE TWO ASCII
* CHARACTERS WILL BE PLACED. B POINTS TO THE MOST SIGNIFICANT
* PORTION OF THE ANSWER.
*
*****
BYTE$TO$ASCII: PROCEDURE (A,B,C) PUBLIC;
  DCL (A,B,C) ADDRESS,
  ENTRY BASED A BYTE,
  OUT1 BASED B BYTE,
  OUT2 BASED C BYTE,
  TEMP BYTE;
  TEMP = SHR(ENTRY,4);
  IF TEMP <= 9 THEN OUT1 = TEMP + 30H;
  ELSE OUT1 = TEMP + 37H;
  TEMP = ENTRY AND 00FH;
  IF TEMP <= 9 THEN OUT2 = TEMP + 30H;
  ELSE OUT2 = TEMP + 37H;
  END BYTE$TO$ASCII;
/

```


BASICS

GET\$BYTE

```

/*****
*
* GET$BYTE:
* PROCEDURE USED TO OBTAIN A DECIMAL NUMBER BETWEEN 0 AND 255 FROM
* THE CRT.
*
* PARAMETERS:
* - DIGITS. - INDICATES THE NUMBER OF DIGITS DESIRED. MUST BE LESS THAN
* OR EQUAL TO 3, ALTHOUGH NO CHECK OF THIS IS MADE.
*
* USAGE:
* TYPED PROCEDURE THAT WILL RETURN A DECIMAL DIGIT OBTAINED FROM
* THE CRT AND REPRESENTABLE IN 3 OR LESS DIGITS.
* THE VALUE RETURNED WILL ALWAYS BE LESS THAN 255.
*
*****/
GET$BYTE: PROCEDURE(DIGITS) BYTE PUBLIC;
DECL (NUMBER,DIGITS,CHAR,COUNT) BYTE;
NUMBER, COUNT = 0;
DO WHILE DIGITS > 0;
  CHAR = CRT$READ;
  DO WHILE ((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> RUB)) OR
    ((CHAR = RUB) AND (COUNT = 0));
    CALL SEND$BEL;
    CHAR = CRT$READ;
  END;
  IF CHAR <> RUB
  THEN DO;
    CALL CRT$WRITE (CHAR);
    NUMBER = NUMBER*10 + (CHAR - 30H);
    COUNT = COUNT + 1;
  END;
END;

```

BASICS

GET\$BYTE

```
DIGITS = DIGITS - 1;  
END;  
ELSE DO;  
  NUMBER = NUMBER/10;  
  CALL SEND$BS;  
  COUNT = COUNT - 1;  
  DIGITS = DIGITS + 1;  
END;  
END;  
RETURN NUMBER;  
END GET$BYTE;
```

BASICS

GET\$ADDRESS

```

/*****
*
* GET$ADDRESS:
*   PROCEDURE USED TO OBTAIN A DECIMAL NUMBER BETWEEN 0 AND 65535 FROM
*   THE CRT.
*
* PARAMETERS:
*   - DIGITS - INDICATES THE NUMBER OF DIGITS DESIRED. MUST BE LESS THAN
*   OR EQUAL TO 5, ALTHOUGH NO CHECK OF THIS IS MADE.
*
* USAGE:
*   TYPED PROCEDURE THAT WILL RETURN A DECIMAL VALUE OBTAINED FROM THE
*   CRT AND REPRESENTABLE IN 5 OR LESS DIGITS. THE VALUE RETURNED WILL
*   ALWAYS BE LESS THAN 65536.
*
*****/
GET$ADDRESS: PROCEDURE(DIGITS) ADDRESS PUBLIC;
DECL CHAR, DIGITS, COUNT) BYTE,
    NUMBER ADDRESS;
    NUMBER, COUNT = 0;
    DO WHILE DIGITS > 0;
        CHAR = CRT$READ;
        DO WHILE ((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> RUB)) OR
            ((CHAR = RUB) AND (COUNT = 0));
        CALL SEND$BEL;
        CHAR = CRT$READ;
    END;
    IF CHAR <> RUB
    THEN DO;
        CALL CRT$WRITE(CHAR);
        NUMBER = NUMBER*10 + (CHAR - 30H);
    
```

BASICS

GET\$ADDRESS

```
    COUNT = COUNT + 1;  
    DIGITS = DIGITS - 1;  
    END;  
    ELSE DO;  
        CALL SEND$BS;  
        NUMBER = NUMBER/10;  
        COUNT = COUNT - 1;  
        DIGITS = DIGITS + 1;  
        END;  
    END;  
    RETURN NUMBER;  
    END GET$ADDRESS;
```


BASICS

GET\$STRING

```

/*****
*
* GET$STRING:
* PROCEDURE USED TO OBTAIN A STRING OF 'NUMBER' CHARACTERS FROM THE
* CRT.
*
* PARAMETERS:
* - NUMBER - NUMBER OF CHARACTERS DESIRED.
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED
* TO BE PLACED.
*
* USAGE:
* UNTYPED PROCEDURE. THE ONLY CHARACTERS THAT CAN BE ACCEPTED FROM
* THE CRT ARE: NUMBERS, UPPER AND LOWER CASE ALPHABETICS, ALTHOUGH
* ALL LOWER CASE ALPHABETICS WILL BE CONVERTED INTO UPPER CASE.
*
* *****/
GET$STRING: PROCEDURE(A,NUMBER) PUBLIC;
  DCL A ADDRESS,
  COUNT = 0;
  DO WHILE NUMBER > 0;
    CHAR = CRT$READ;
    DO WHILE ((CHAR < 30H) OR (CHAR > 7AH) OR ((CHAR > 39H) AND (CHAR < 41H))
    OR ((CHAR > 5AH) AND (CHAR < 61H))) AND (CHAR <> RUB)) OR
    ((CHAR = RUB) AND (COUNT = 0));
    CALL SEND$BEL;
    CHAR = CRT$READ;
  END;
  IF (CHAR >= 61H) AND (CHAR <= 7AH)
  THEN CHAR = CHAR - 20H;

```

GET\$STRING

BASICS

```
IF CHAR <> RUB
THEN DO:
  CALL CRT$WRITE (CHAR);
  A = A + 1;
  NUMBER = NUMBER - 1;
  COUNT = COUNT + 1;
END;
ELSE DO:
  CALL SEND$BS;
  A = A - 1;
  NUMBER = NUMBER + 1;
  COUNT = COUNT - 1;
END;
END;
END GET$STRING;
```

BASICS

PUT\$NUMBER\$BUFFER

```

/*****
*
* PUT$NUMBER$BUFFER:
* THIS PROCEDURE IS USED TO OBTAIN AN SPECIFIED NUMBER OF NUMERIC CHARACTERS
* FROM THE CRT, AND TO PUT THEM IN A GIVEN MEMORY LOCATION.
*
* PARAMETERS:
* - NUM - NUMBER OF NUMERIC CHARACTERS DESIRED.
* - A - POINTER TO A MEMORY LOCATION IN WHICH THE STRING IS DESIRED TO
* BE PLACED.
*
*****/
PUT$NUMBER$BUFFER: PROCEDURE(NUM,A) PUBLIC;
DCL A ADDRESS,
COUNT = 0;
DO WHILE NUM > 0;
CHAR = CRT$READ;
DO WHILE ((CHAR < '0') OR (CHAR > '9')) AND (CHAR <> RUB) OR
((CHAR = RUB) AND (COUNT = 0));
CALL SEND$BEL;
CHAR = CRT$READ;
END;
IF CHAR <> RUB
THEN DO;
CALL CRT$WRITE(CHAR);
TEMP = CHAR;
A = A + 1;
NUM = NUM - 1;
COUNT = COUNT + 1;
END;
*****/

```

PUT\$NUMBER\$BUFFER

BASICS

```
ELSE DO;  
  CALL SEND$BS;  
  A = A - 1;  
  NUM = NUM + 1;  
  COUNT = COUNT - 1;  
END;  
END;  
END PUT$NUMBER$BUFFER;
```

END BASICS;

LIST OF REFERENCES

1. Babin, O. P. and Seaman, R. S., A Microcomputer Based Plasma Display System, M. S. Thesis Naval Postgraduate School, Monterey, California, 1978.
2. Elite 2500 Instruction Manual, Datamedia Corporation, 7300 N. Crescent Blvd., Pennsauken, N. J., 08110.
3. Kerns, K. H. and Cooper, R. S., A Microcomputer Solution to Maneuvering Board Problems, M.S. Thesis Naval Postgraduate School, Monterey, California, 1973.
4. Hastings, C.Jr., Approximations for Digital Computers, Princeton, New Jersey, Princeton University Press, 1955.
5. Intellec 800 Microcomputer Development System Operator's Manual, Intel Corporation, Santa Clara, California, 1975.
6. Intellec Microcomputer Development System Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1976.
7. ISIS-II PL/M 80 Compiler Operator's Manual, Intel Corporation, Santa Clara, California, 1976.
8. ISIS-II System User's Guide, Intel Corporation, Santa Clara, California, 1976.
9. PL/M 80 Programming Manual, Intel Corporation, Santa Clara, California, 1976.
10. SBC 310 High-Speed Mathematics Unit Hardware Reference Manual, Intel Corporation, Santa Clara, California, 1977.
11. Newman, W. M. and Sproull, R. F., Principles of Interactive Computer Graphics, McGraw-Hill Computer Science Series, 1973.
12. Operation Specialist 3 & 2 (UNCLASSIFIED), Naval Education and Training Command, Rate Training Manual and Nonresident Career Course, NAVEDTRA 10144-C, 1975.
13. Plasma Display Set Technical Manual Vol. I, Science Applications Inc., San Diego, California, 1976.
14. Plasma Display Set Technical Manual Vol. II, Science Applications Inc., San Diego, California, 1976.

15. Scheid, F., Theory and Problems of Numerical Analysis,
Schaum's Outline Series, McGraw Hill, 1968.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2. Fleet Support Office Naval Ocean Systems Center San Diego, California 92152	1
3. Dr. William A. Von Winkle Associate Technical Director for Technology Naval Underwater Systems Center New London, Connecticut 06320	1
4. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
5. Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
6. LCDR Stephen T. Holl, USN, Code 52H1 Department of Computer Science Naval Postgraduate School Monterey, California 93940	3
7. Professor George A. Rahe, Code 52Ra Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
8. Associate Professor Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	3

9. LCDR Antonio Luiz S. Goncalves, Brazilian Navy 6
Rua Prudente de Moraes, 660 apto. 202
Ipãnema, Rio de Janeiro 20000
RJ, BRAZIL
10. LT(JG) Javier E. De la Cuba Bravo, Peruvian Navy 6
Direccion de Instruccion de la Marina
Ministerio de Marina
Ave. Salaverry S/N
Lima, PERU
11. LCDR Ken Clare, USN 1
Code 112 NAVDAC
Washington, D.C. 20374
12. Dr. Joel Trimble 1
Office of Naval Research
Arlington, Virginia 22217
13. LT Mark S. Moranville, USN, Code 52Mi 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93940